You Build It You Run It

Playbook

Bethan Timmins
Steve Smith

PLAYBOOKS.EQUALEXPERTS.COM/YOU-BUILD-IT-YOU-RUN-IT

EQUAL EXPERTS

Welcome to the Equal Experts You Build It You Run It playbook.

This summarises our thinking on when, why, and how to implement the **You Build It You Run It** operating model.

You Build It You Run It is an operating model in which product teams build, deploy, operate, and support their own digital services

This playbook is based on first-hand experiences from across the **Equal Experts network**.

Equal Experts has worked with hundreds of organisations. Our customers often ask us for advice on how to improve their deployment throughput, service reliability, and learning culture.

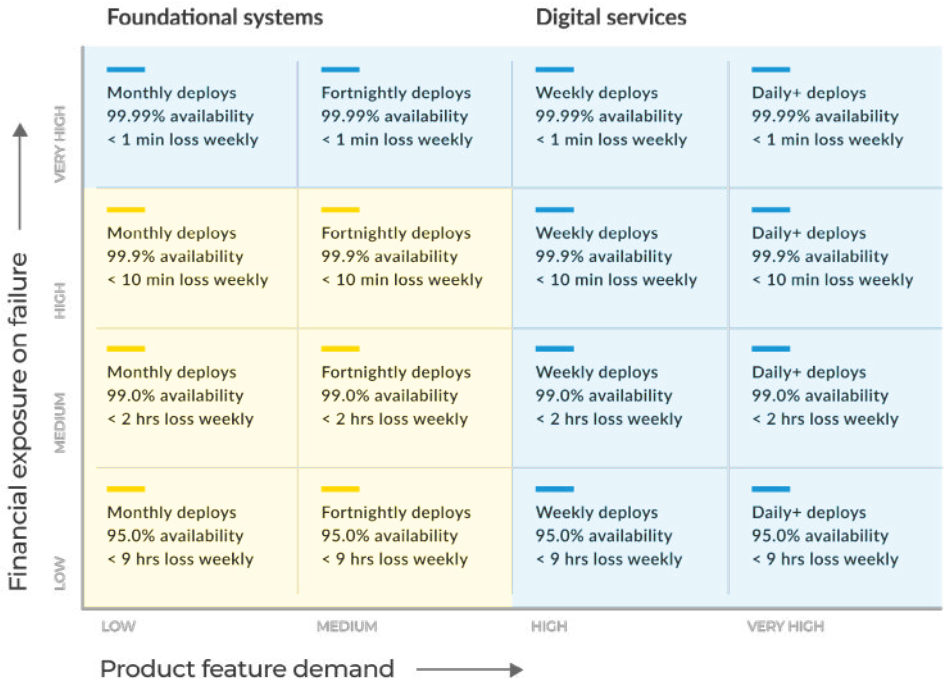
You Build It You Run It is a cost-effective way to achieve those goals together:

- *Higher deployment throughput.* Weekly deployments, or more frequent.
- *Greater service reliability.* 99.9% availability, or higher.
- *Continuous learning culture.* Constantly generating insights and implementing improvements.

This playbook contains the principles, practices, and pitfalls we associate with You Build It You Run It for digital services. The first part describes the traditional operating model of a central operations team, which we call Ops Run It. It also explains why using on-call product teams doesn't mean the end of a central operations team.

A central operations team isn't the right choice for digital services, yet it remains the right choice for self-hosted foundational systems (COTS and custom integrations), which have lower deployment throughput and service reliability needs.

Operating model selector



A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

You Build It You Run It

We advocate for You Build It You Run It and Ops Run It co-existence in a hybrid operating model, which can be framed in terms of deployment throughput, service reliability, and learning culture.

We've open-sourced this playbook under a Creative Commons license, and we'd welcome contributions to improve it.

Authors

The authors of this playbook are:

Bethan Timmins

Managing Director

Equal Experts

Australia & New Zealand

Steve Smith

Principal Consultant

Equal Experts

UK

Bethan Timmins is a Principal Consultant and Managing Director for Equal Experts A&NZ, and is responsible for the growth and success of Equal Experts within Australia and New Zealand. With over 20 years experience working in software delivery across multiple sectors, her delivery experience has a focus on:

- understanding and articulating the product being built
- improving on the process to feed a delivery team
- increasing the flow and delivery of ideas through to customers

Bethan has played key roles in multiple digital transformation programmes globally, including one of the largest cloud migration projects in the UK government. Bethan was a finalist in the Women in IT Awards 2018 for her expertise in transformation, and is an advocate for getting more women into senior Tech roles.

Steve is a Principal Consultant and Practice Lead at Equal Experts, who advises clients worldwide on Continuous Delivery, operating models, and reliability.

Steve has played key roles in multiple digital transformation programmes. He was a leader on a 60 team/600 microservices programme for a UK government department with £500Bpa revenue. He also implemented a Continuous Delivery and operability strategy for 30 teams/100 microservices in a high street retailer with £2Bpa online revenue. He's also written multiple books, such as ***Measuring Continuous Delivery***.

EQUAL EXPERTS

Introduction

At Equal Experts, we help organisations to achieve sustainable innovation. If your organisation has a similar ambition, you'll likely need improvements in:

- ***Deployment throughput.*** Digital services need to constantly evolve, to meet your changing customer needs in a competitive landscape. ***Accelerate*** by Dr. Nicole Forsgren *et al* found organisations with a high deployment throughput were twice as likely to exceed profitability, market share, and productivity expectations.

- ***Service reliability.*** Digital services need to be consistently reliable, to protect your revenue, costs, and brand reputation. A **Fortune 1000 survey** by IDC found the average cost of a critical failure was between $0.5M and $1M per hour.

- ***Learning culture.*** To foster high-performing teams, a culture of continuously creating insights and implementing improvements is required. ***A typology of organisational cultures*** by Ron Westrum demonstrates that a stronger culture results in higher levels of trust, collaboration, and higher quality decision-making.

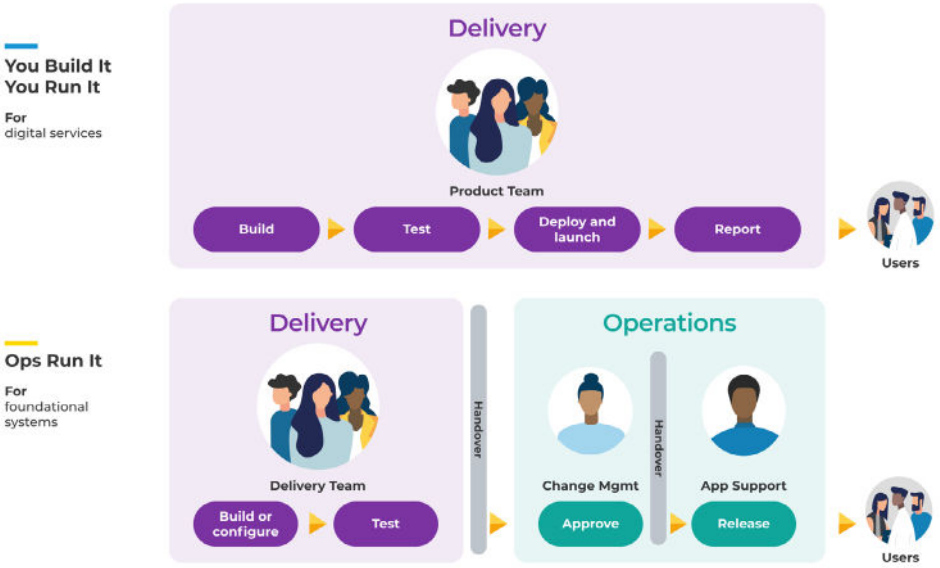
These improvements are essential if you want to optimise your customer experience, achieve delivery excellence, and minimise your operational costs. Robert Charette estimated in ***Inside the hidden world of legacy IT systems*** that 75%, or $26.25 trillion of worldwide IT spending 2010-2020 was on operational expenditure.

Many of our customers use the traditional operating model, of one or more operations teams doing deployments and providing production support. We call this **Ops Run It**. We believe Ops Run It can't achieve the standards of deployment throughput, service reliability, and learning culture required for digital service management. This has been validated by Charles Betz of Forrester Research, who warns in ***The Future of Technology Operations*** that "the old guard operating practices of ITIL, COBIT, PMBoK, and CMMI are rapidly losing relevance", and "a plan/build/run lifecycle is no longer suitable".

You Build It You Run It is a modern operating model. It's the de facto name for on-call product teams owning all aspects of their custom digital services, from inception to decommission. They launch to live traffic, monitor customer behaviours, and respond to production incidents themselves. You Build It You Run It transforms technology operations from reactive ticket management to proactive continuous improvement.

We recommend You Build It You Run It for higher demand digital services. Adopting You Build It You Run It means comprehensive changes for people, processes, and technology. It requires the creation of cross-functional teams who are responsible for development, testing, and production support of digital services. It means redefining roles, streamlining service management processes, and building a fully automated toolchain from deployment pipeline to incident management.

Deployment throughput in You Build It You Run It



Service reliability in You Build It You Run It

You Build It You Run It

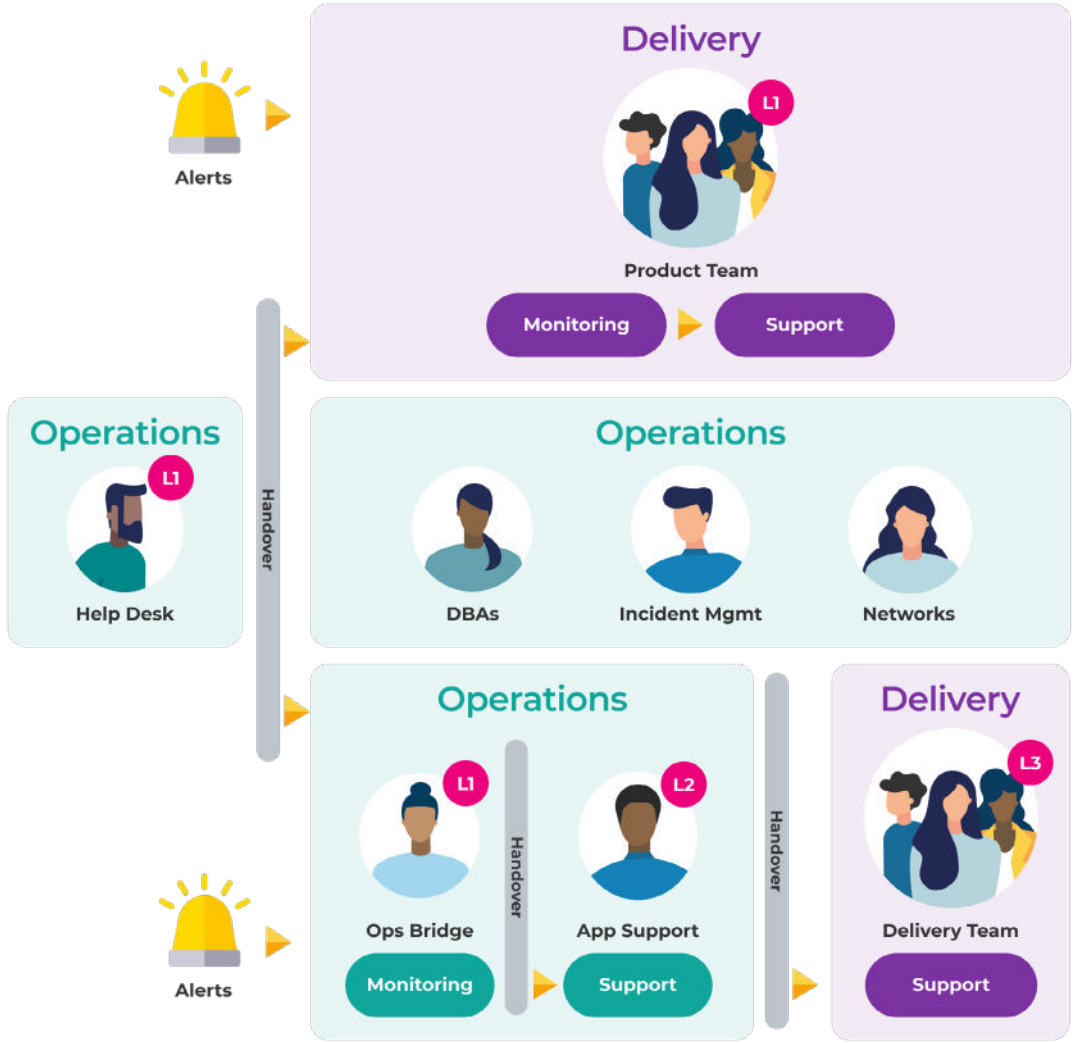
For
digital services

Operational Enablers

For
everything

Ops Run It

For
foundational systems



Learning culture in You Build It You Run It

You Build It You Run It

For
digital services

Operational Enablers

For
everything

Ops Run It

For
foundational systems

We still advise using Ops Run It for lower demand, self-hosted foundational systems such as COTS applications and custom integrations. A central operations team remains a cost-effective option 'where demand for product features is low'.

This means that modern technology operations equates to a hybrid operating model - You Build It You Run It for digital services, and Ops Run It for foundational systems.

What is in this playbook

This playbook compares You Build It You Run It to Ops Run It. The comparison is in terms of deployment throughput, service reliability, and learning culture.

Deployment throughput is the production output for a service. It is expressed in terms of deployment frequency and deployment lead time. Creating a fully automated deployment pipeline and modernising ways of working accelerates the rate of production deployments. See ***Measuring Continuous Delivery*** by Steve Smith.

Service reliability is the ability of a service to function without failure. This is a narrow definition that omits functional correctness and customer experience. Service reliability can be expressed as an availability level, and time to restore availability. Improving the operability of a service increases its availability, by creating sources of adaptive capacity. See ***On adaptive capacity in incident response*** by John Allspaw *et al*.

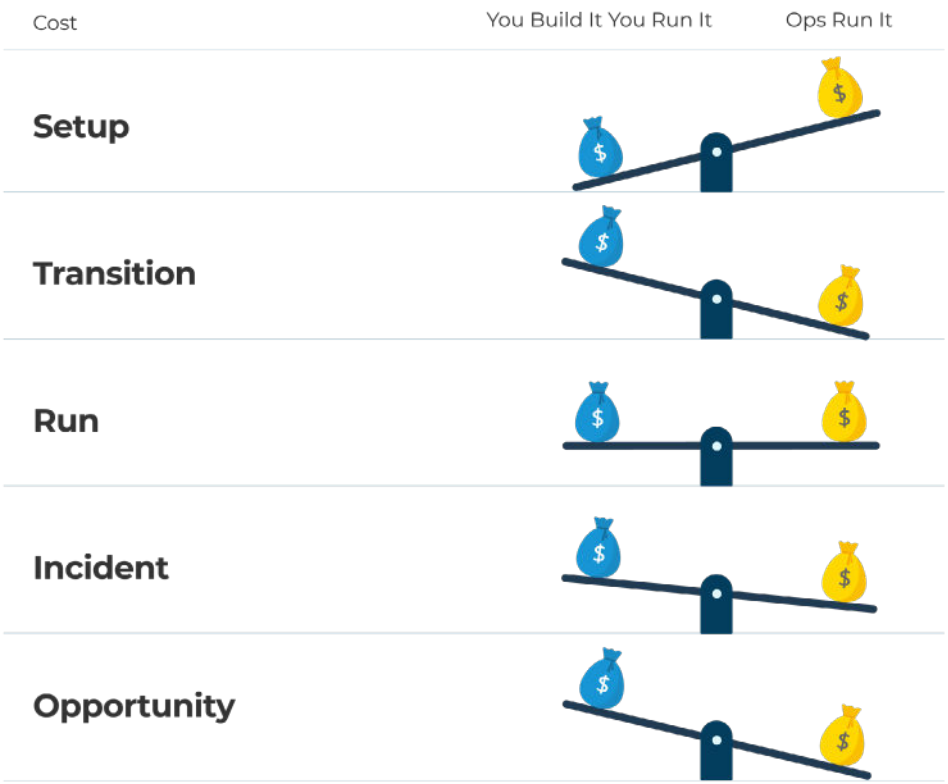
A learning culture is how people acquire new knowledge and skills, within the shared, implicit assumptions underpinning their social behaviours. It's a cycle of never-ending improvement, based on generating insights and implementing improvements from deployment and availability problems. A strong learning culture empowers people to experiment, make more informed decisions, and collaborate on shared organisational goals. See ***Continuous learning as a tool for adaptation*** by Nora Jones.

This table summarises the characteristics of You Build It You Run It and Ops Run It.

	YOU BUILD IT YOU RUN IT	OPS RUN IT
DEPLOYMENT THROUGHPUT		
Deployment frequency	Daily+ to weekly	Fortnightly to monthly
Deployment lead time	Minutes to days	Days to months
Total cost of ownership % estimate	Low to medium	Medium to high
SERVICE RELIABILITY		
Developer operability incentives	Daily+ to weekly	Fortnightly to monthly
Availability level	Minutes to days	Days to months
Time to restore availability	Daily+ to weekly	Fortnightly to monthly
Total cost of ownership % estimate	Low to medium	Medium to high
LEARNING CULTURE		
Learning predisposition	Daily+ to weekly	Fortnightly to monthly
Time to generate insights	Minutes to days	Days to months
Time to implement actions	Daily+ to weekly	Fortnightly to monthly
Total cost of ownership % estimate	Low	Medium
STANDARDS		
Supported standards	ITIL v3 ITIL v4 PCI-DSS SOX	COBIT ITIL v3 ITIL v4 PCI-DSS SOX

We've established relative cost estimates for deployment throughput, service reliability, and learning culture in Ops Run It and You Build It You Run It. These estimates are based on our shared experiences of working with on-call product teams and central operations teams, in many different organisations. Detailed analyses are available in What is Ops Run It and What is You Build It You Run It. We believe Ops Run It has much higher opportunity costs, whereas You Build It You Run It has higher setup costs, and the run costs are similar.

Relative cost estimates



Who is this playbook for

We've written this playbook for CxOs, product managers, delivery managers, and operations managers. We're looking to help people answer questions such as:

- Why is it so difficult to accomplish weekly deployments and/or 99.9% availability with a central operations team?
- What are the key characteristics of on-call product teams, in terms of deployment workflow, availability restoration, and learning from failures?
- How can the cost effectiveness of on-call product teams be measured, and compared with a central operations team?
- What are the advantages and disadvantages of on-call product teams, compared to a central operations team?
- What mistakes can be made when starting out with on-call product teams, and how can we avoid them?

We've answered these questions in a way that will benefit a majority of readers, while recognising that context always matters.

Readers will notice us favouring Continuous Delivery and Operability over DevOps and Site Reliability Engineering (SRE). See *The value of operability* by Dan Mitchell, and *What you should (and probably shouldn't) try from SRE* by Steve Smith and Ali Lotia.

Ops Run It

What is Ops Run It

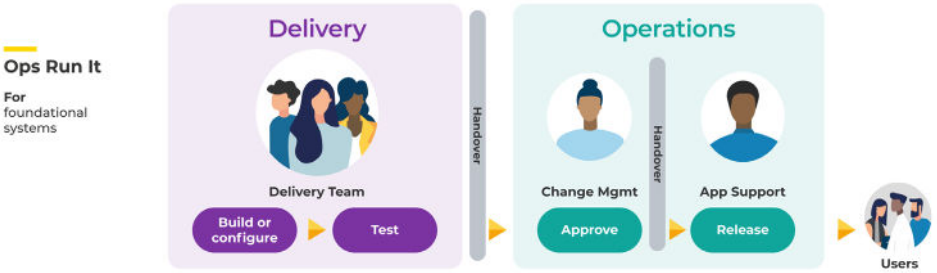
Many organisations have an IT department with segregated Delivery and Operations functions. Delivery teams in the former are responsible for building services, and operations teams in the latter are responsible for running services. We call this Ops Run It. We often find it's entwined with long-established IT management standards such as ITSM and ITIL v3.

Ops Run It has been a de facto operating model for decades, and was codified by the COBIT management framework in 1996. COBIT recommended functionally-oriented teams of specialists, working within separate Plan, Build, and Run phases. That was justified by the unavoidably high compute and transaction costs for on-premise software in the 1990s. That pre-Internet rationale doesn't hold true today.

Ops Run It creates a hard divide between Delivery and Operations, powered by radically different incentives. Delivery teams are short-lived, and project-based. They're told to move fast and achieve their deliverables. Operations teams are long-lived, and told to move carefully and provide reliability.

Here's an example of what Ops Run It looks like. It's broken down in terms of deployment throughput, service reliability, and a reactive culture. It's a composite example drawn from multiple customers, but it's still important to remember that every organisation is different, and every Ops Run It implementation is different.

Deployment throughput in Ops Run It



Ops Run It teams vary between organisations. Your application support team could be called application operations. Your change management and application support teams could be augmented by a service introduction team, who assess operational readiness prior to live launch.

A delivery team performs automated functional testing and exploratory testing. When they want a production deployment to happen, they need to follow a service management process owned by the operations teams:

1. *Delivery team requests change.* The delivery team adds a ticket into the change management queue, in a workflow system such as ServiceNow. In ITIL v3, this is classified as a normal change request.

2. *Change management team reviews change.* When the change management team picks up the ticket, they approve or reject the change request in a Change Advisory Board (CAB) meeting, based on their knowledge of other in-flight changes.

3. *Change management team requests deployment.* If the change is approved, the change management team schedules a ticket into the application support queue.

4. *Application support team performs deployment.* When the application support team picks up the ticket, they use a tool such as Jenkins or GitLab CI to orchestrate the deployment. A deployment can happen out of hours, if downtime is required.

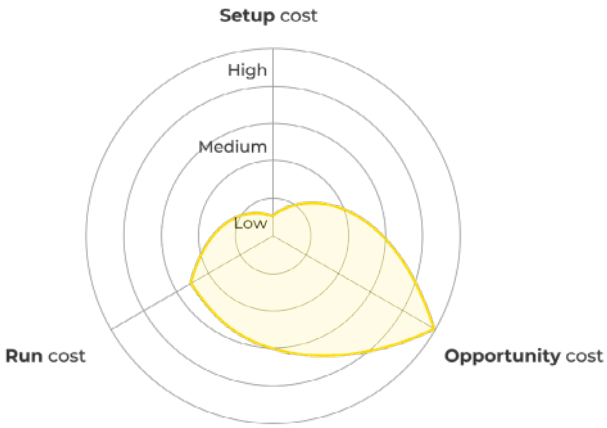
5. *Delivery team performs post-deployment validation.* The delivery team does some post-deployment smoke testing, before the application support team accepts ownership of the new live service.

We've occasionally seen a deployment throughput variant of Ops Run It, in which delivery teams can perform some classes of deployments themselves. It requires a high degree of automated functional testing, and a fully automated deployment pipeline including change management approvals and production deployments. Change requests are pre-approved for small, regular changes. Large, irregular changes still require a CAB meeting and deployment by the application support team. In ITIL v3, this is the difference between standard and normal change requests.

This variant depends on a high trust environment, with a strong culture of collaboration. It's heavily dependent on senior leaders in Delivery and Operations functions sharing a sense of urgency, and teams working hard to recognise their similarities rather than their differences. We've only seen this work in a few mid-sized organisations, and we've not seen it scale to many teams.

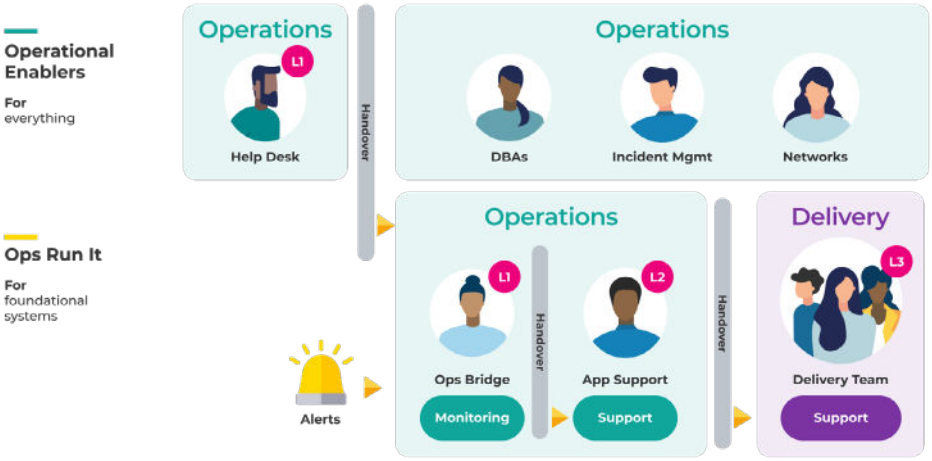
Deployment throughput costs in Ops Run It

We've attributed to Ops Run It deployment throughput costs an estimate of how much they usually contribute to the total cost of ownership. Per deployment costs are charged to operational expenditure (opex) rather than capital expenditure (capex), as the operations teams are centralised and distinct from any particular service.



COST TYPE	FREQUENCY	DESCRIPTION	IMPACT	TCO % ESTIMATE
Setup cost	One-off	Launch costs incurred in • Change management team time for setup • Application support team time for deployment setup	Capex cost	Low
Opportunity cost	Per feature	Can be measured as the cost of delay between product feature readiness (approval) and its launch. Potential revenue lost, missed customer opportunities due to: • Delay waiting for CAB meeting • Delay waiting for production deployment and launch	Lost revenue, capex or opex cost increase	High
Run cost	Per deployment	Deployment costs incurred in completing the deployment • Time to produce handover documentation for change management team • Time to produce handover documentation for application support team • Change management team time for CAB • Change reporting licenses • Time for performing deployments, and rolling back failed deployments	Opex cost	Medium to high

Service reliability in Ops Run It



Service reliability involves a multi-level hierarchy of teams - an L1 operations bridge team, an L2 application support team, L3 delivery teams. There are also operational enablers who can be called upon for assistance, such as incident management and DBAs. This setup varies between different organisations:

- Your customer self-service capabilities could be classified as L0.
- Your help desk team could be called customer services.
- Your operations bridge team could be called the operations centre.
- Your application support team could be called application operations.
- Your help desk and operations teams could be a single L1 team.
- Your operations bridge and application support teams could be a single L2 team.

The L1 help desk team receives customer complaints about service availability. They attempt to use documentation to resolve high volume, straightforward issues.

The operations bridge and application support teams offer 24/7 production support. They are responsible for day-to-day work in availability protection and availability restoration. They track their costs centrally, and make them visible to senior leaders. They report into the senior manager in the Operations function, who is accountable for all aspects of reliability.

Application support analysts can modify the below facets of a digital service:

- Alert definitions
- Configuration
- Data
- Deployments
- Infrastructure definitions
- Monitoring dashboards

For governance, the senior operations manager is accountable for all aspects of digital service reliability. The operations bridge and application support teams share responsibility for availability protection, and availability restoration.

Availability protection in Ops Run It

The operations bridge team proactively monitors service telemetry for abnormalities. This is sometimes known as 'eyes on glass'. They observe service health checks, logs, and metrics plumbed into different dashboards, in telemetry tools such as AppDynamics or New Relic. Their dashboards are dependent on monitorable events, which are pre-defined by delivery teams within the services. Changes to the monitorable events are rare.

The application support team proactively updates digital services and their runtime environments, using tools like Red Hat Satellite. They add infrastructure capacity, make configuration changes, apply data fixes, and update dashboards, to reduce drift and prevent abnormal conditions. This is labour intensive toil, and the workload can increase dramatically for a peak business event.

Delivery teams are usually unaware of the heroics performed by operations teams in availability protection. There are some scenarios in which they can be temporarily promoted to L1 availability protection, alongside operations teams:

- *Hypercare.* Support for a few weeks after service launch. Also known as a warranty period.
- *Peak support*. Support for a peak business event, such as Black Friday. Also known as enhanced support.

Availability restoration in Ops Run It

When availability for a service falls below its availability target, an incident is declared and efforts are made to restore availability. In ITIL v3 this is overseen by an incident manager. The vast majority of incidents are expected to be resolved by the L1 help desk and operations bridge teams, with a minority reaching L2 application support. Very few are expected to reach the L3 delivery teams.

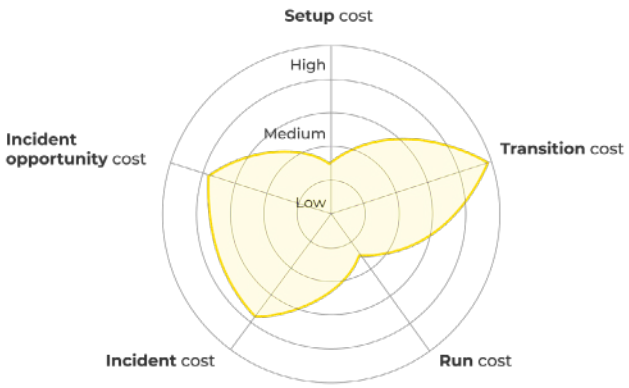
The operations bridge team receives service alerts that warn of availability errors. They raise a ticket in the incident tracking system and record all the service telemetry data at hand. They can try to restore availability with restart scripts. If unsuccessful, they reassign the ticket and escalate to the application support team.

The application support team receives notifications of service unavailability from their ticket queue. They diagnose the causes of unavailability via heuristics, expertise, and service telemetry data. They attempt to restore availability via additional infrastructure, data fixes, config changes, rollbacks, and telemetry changes. If the service remains unavailable, they locate the delivery team that owns the service and escalate the incident to them.

Delivery teams receive notifications of service unavailability from the application support team, when availability cannot be restored by operations teams alone. The notification is often informal, and urgent. One or more delivery team developers collaborate with the application support team to clarify the causes of service unavailability, and restore the service via whatever means necessary. This can include deployments of emergency code changes.

Service reliability costs in Ops Run It

For Ops Run It reliability costs, we're assuming on-call standby costs and on-call callout are both present, although organisations can have very different remuneration models.



COST TYPE	FREQUENCY	DESCRIPTION	IMPACT	TCO % ESTIMATE
Setup cost	One-off	Launch costs incurred in • License purchases • Application support team time for telemetry install • Application support team time for schedule agreement	Capex cost	Low
Transition cost	One-off	Launch costs incurred in • Application support team time for handover plan agreement • Delivery team time for implementing operational acceptance criteria	Capex cost	High
Run cost	Ongoing	Regular costs incurred in application support team time for • Applying data fixes • Making configuration changes • Adding infrastructure capacity • Performing rollbacks • Updating telemetry tools • On-call standby out of hours	Opex cost	Low to medium
Incident cost	Per incident	Incident response costs incurred in application support team time for • Applying data fixes • Making configuration changes • Adding infrastructure capacity • Performing rollbacks • Updating telemetry tools • On-call callout out of hours	Opex cost	Medium to high
Opportunity cost	Per incident	Can be measured as the cost of delay between incident start and finish. Caused by service unavailability, missed opportunities with customers, and delays in further feature development	Revenue loss and costs incurred	Medium to high

In Ops Run It, we often observe operations managers under pressure to reduce their opex spend. The common countermeasures are:

- Outsource L1 and L2 support. Reduce on-call standby rates, on-call callout rates, and operations team payroll costs by outsourcing the service desk, operations bridge, and/or application support teams to a third party managed service. The contract is likely to be multi-year, and could include service credits linked to an incident count threshold.

- Do not pay for L3 support. Avoid on-call standby rates and on-call callout rates by ensuring delivery team developers are not paid for L3 best efforts on-call. There can be unofficial compensation as informal time off in lieu.

These countermeasures have trade-offs, and they contribute to **Ops Run It drawbacks** as well.

Reactive culture in Ops Run It

We refer to the Ops Run It reactive culture as predisposed to 'rush to fix'. When there is an incident of consequence, such as a failed deployment or a loss of service availability, the priority is to fix the underlying problem and return to the prior state. In ITIL v3, this is considered part of problem management, and overseen by a problem manager. Your customer self-service capabilities could be classified as L0.

Find the root cause

After an incident, a problem manager schedules a root cause analysis session within days or weeks. The purpose is to find the root cause of the failure, and implement any fix actions afterwards.

Attendees in a root cause analysis session include the application support analysts and incident managers that participated in incident response. The problem manager facilitating the session can invite delivery team developers if the incident was escalated to an L3 callout.

The session format is:

- Share incident observations and service telemetry data.
- Capture a financial impact assessment, including revenue losses and costs incurred.
- Document a single timeline of events from incident start to incident finish.
- Agree on a single, perceived human or technical error that caused the failure.
- Establish a series of fix actions with the aim of permanently fixing the problem.

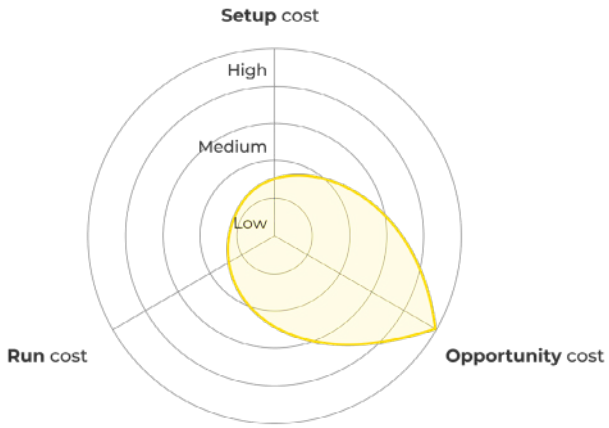
The problem manager summarises the session in a root cause analysis document. This is shared with session attendees, and depending on the circumstances perhaps the delivery and operations teams as well.

Implement fix actions

After the session, the problem manager records the root cause as a problem in the company workflow system, and the recommended fix actions. Documentation fixes are handed over to the application support team, such as amending training procedures or updating a service runbook with a workaround for the problem. Code and configuration fixes are handed over to the delivery team(s).

Reactive culture costs in Ops Run It

Ops Run It learning culture costs are incurred each time someone has an idea on how to improve a digital service.



COST TYPE	FREQUENCY	DESCRIPTION	IMPACT	TCO % ESTIMATE
Setup cost	One-off	Launch costs incurred in application support team training for root cause analysis	Opex cost	Low to medium
Insight opportunity cost	Per insight	Can be measured as the cost of delay between generating an insight and implementing the related actions. Potential revenue lost and/or costs incurred due to service unavailability, missed opportunities with customers, and delays in feature development.	Lost revenue Capex or opex cost	High
Insight implementation cost	Per insight	Costs incurred in implementing the actions linked to an insight. This could include adding infrastructure capacity, code fixes, documentation updates, etc.	Capex or opex cost	Low to medium

Benefits of Ops Run It

We've split Ops Run It benefits into realised versus unrealised benefits, i.e. actual versus on paper. In our experience, many of the theoretical benefits of a central operations team simply don't happen for digital services.

Realised

These are the few benefits for digital services we expect to actually see in practice.

Deployment throughput

We don't usually see any deployment throughput benefits. For a majority of organisations, **Ops Run It can't achieve weekly or daily deployments**. Some small or medium-sized organisations may overcome this, if they implement the **Ops Run It deployment throughput variant** and delivery teams self-service some deployments. However, it's unusual to see that work well.

Service reliability

We see some benefits, related to cost management and governance:

- *Low setup costs.* Ops Run It has a **low implementation cost**, as it's so well-established in the IT industry. There's plenty of training courses, online reading materials, and vendor tools that are compatible with multi-level production support.
- *Low run costs.* Ops Run It has a **low run cost**. The help desk, operations bridge, and application teams can centrally manage the availability of tens of digital services, plus self-hosted COTS applications and custom integrations.
- *Run cost reduction options.* Run cost can be **lowered by outsourcing**, as the general skills and technical expertise for help desk, operations bridge, and application support teams are in a worldwide abundance.
- *Straightforward governance.* Accountability, responsibilities, and visibility are all easy to understand. The senior manager in the Operations function is accountable for all aspects of reliability.

For foundational systems, these benefits can outweigh the **unrealised benefits** and **many drawbacks of Ops Run It**. We've seen plenty of organisations outsource one or more operations teams to a third party managed service, to minimise their run costs. However, if an application support team has built up substantial domain expertise with a proven impact on incident resolution times, the case for outsourcing is not as clear cut.

Learning culture

'We see a weak learning culture in Ops Run It, without many benefits. People can learn individually from incidents, but the opportunities for entire teams to glean new knowledge are limited. There's limited new knowledge, and people are stuck in firefighting.

Unrealised

These are Ops Run It benefits for digital services that are often described in theory, yet we rarely see in practice.

Deployment throughput

- *High speed delivery teams.* Insulating delivery teams from production deployments is supposed to accelerate the speed of new product features. Instead, we see large deployments stuck in handovers between delivery and operations teams, a lot of rework when deployments fail, and little positive impact on customer outcomes.

- *Consistent deployment process.* Using the same change management and application support teams is intended to create a consistent, well-understood process for deployments. We often see a highly variable process, which depends on personal relationships or management escalations for deployments to happen when expected.

Service reliability

Availability protection:

- *Informative telemetry*. An operations bridge team is theoretically able to amend logs, metrics, dashboards, and alerts over time, to increase their understanding of operating conditions. We consistently see growing telemetry datasets with a low information value.

- *Comprehensive documentation.* Operations teams are supposed to have access to accurate documentation, to help them protect service availability. We invariably find a lack of up to date documentation from delivery teams.

Availability restoration:

- *Fast alert acknowledgement.* Having operations bridge and application support teams always on-call is meant to minimise the time between alert and acknowledgement. We've seen organisations where it takes up to twenty minutes for an application support analyst to become aware of an alert.

- *Fast incident resolution.* Routing an operations bridge team to most incidents and a skilled application support team to the remainder is supposed to minimise the time between alert and acknowledgement. We usually see a time to resolution greater than one hour.

- *Few L2 callouts.* Using an L1 operations bridge team to restart services and follow runbooks is intended to minimise L2 callouts. We nearly always see a high percentage of alerts escalated to L2 callouts, for diagnosis and availability restoration.

- *Low incident costs.* Expecting an operations bridge team to handle most alerts creates an expectation that incident costs will be flattened. We often observe a majority of alerts escalated to the application support team, a slow time to resolve incidents, and much higher incident costs than anticipated.

We suspect these unrealised benefits are based on assumptions such as service unavailability is rare, incident count is controllable, and most issues are easily resolved. Those assumptions are entirely wrong, and it's unlikely these benefits can be achieved. See ***How Complex Systems Fail*** by Dr. Richard Cook.

Learning culture

- ***Informative insights.*** A root cause analysis session concentrates on a single human or technical error. We regularly observe multiple contributing factors to a production incident being silently ignored.
- ***Broad insight dissemination.*** A root cause analysis document is supposed to be shared throughout an organisation, so anyone can benefit from the new knowledge. We rarely see document sharing with operations teams, let alone delivery teams.
- ***Fast time to improve.*** The fix actions from a root cause analysis session are intended to be implemented quickly by operations and delivery teams. We usually find large queues of fix actions that take weeks or months to be prioritised, while operations and delivery teams tackle planned work instead.

We think these unrealised benefits come from the notion that a production incident stems from a single event. Unfortunately, there's no such thing as a single root cause in modern, complex software services. These benefits could be realised if there was a concerted effort to move away from root cause analysis to insight generation, but we rarely see that happen. See ***Each necessary, but only jointly sufficient*** by John Allspaw.

Drawbacks of Ops Run It

Ops Run It has a lot of drawbacks for digital services. They're systemic, and not the fault of any delivery or operations team. They're generally related to the **hard divide between siloed delivery and operations teams**. Countermeasures are sometimes available, albeit with their own costs and complexity.

Ops Run It creates a **diffusion of responsibility**. Product managers are responsible for launching products they don't support, and are incentivised to prioritise product features over operational features. Delivery teams are responsible for building services they don't support, and are incentivised to maximise deployments. Operations teams are responsible for supporting services they don't build, and are incentivised to minimise deployments. This results in a slow time to market, reliability problems, and/or limited learning opportunities.

Due to these drawbacks, Ops Run It can't sustain weekly or more frequent deployments. There's an underinvestment in operability, and a burden on the operations teams grows as the number of delivery teams increases. Eventually, they become a significant organisational bottleneck, through no fault of their own.

Deployment throughput

- *Slow change approvals.* Change requests take hours, days, or weeks to be signed off. In fact, Dr. Forsgren *et al* demonstrated in ***Accelerate*** that not having a change approval process leads to more positive outcomes than an ITIL change process.
  - A change management team can be slow to pick up a change request, if they have one queue for all incoming work and/or it's an ITIL normal change request with a 3-5 day response time.
  - A change management team can be slow to approve a change request, if a CAB meeting requires a business justification, rollback plan, etc. from the delivery team.
- *Slow deployments.* Deployments take hours or days to complete successfully.
  - An application support team can be slow to take on a deployment, if they have one queue for all incoming work and their own operational work to prioritise.
  - An application support team can have to contend with a semi-automated deployment process, intermittent failures, and ambiguous documentation from delivery teams.
  - An application support team can be unable to create an automated deployment pipeline, if ownership of the deployment process is shared with delivery teams.
- *Zero focus on outcomes.* Digital services are created as outputs with no regard for customer outcomes. Delivery teams lack the access to live traffic data necessary to form product hypotheses.
- *High knowledge synchronisation costs*. Knowledge synchronisation takes hours or days between operations and delivery teams. An application support team can require knowledge transfer meetings and/or runbook updates for every deployment.
- *Scheduling conflicts*. Deployments are rescheduled on short notice. An application support team suffers from management escalations and fluctuating priorities, as delivery team stakeholders seek faster results for themselves.
- *Contractual deployment charges*. Traditional supplier contracts discourage an increase in deployments. An outsourced application support team can charge a lower rate for two or three deployments per month, a higher rate for rollbacks, and a much higher rate for more frequent deployments.

These drawbacks add up to high opportunity costs, with potential revenue lost due to the long lead times to launch new product features. The obvious countermeasure is to create a fully automated deployment pipeline, to reduce errors and speed up the deployment process. We'd strongly recommend a deployment pipeline, and caution that it can't tackle most of these drawbacks alone. For example, it can't eliminate the delays incurred on each handoff between teams.

Service reliability

We usually see an overwhelmed application support team. They have to support complex, fragile, poorly-documented digital services, produce miraculous fixes to tough problems, and cope with a high number of L2 callouts.

Availability protection:

- ***Uninformative telemetry.*** Alerts, logs, metrics and dashboards are based on low-level events, which cannot illuminate operating conditions. An operations team can't change monitorable events without digital service changes prioritised by a product manager.
- ***Poor documentation.*** Architecture diagrams, alert threshold guides, configuration settings, and runbooks sourced from delivery teams are either outdated or do not exist.
- ***On-call dissatisfaction.*** Delivery team developers on best efforts on-call out of hours are not compensated for the inconvenience, and disruption to their lives outside of work.
- ***Fragile architecture.*** Services have a large blast radius, an inability to gracefully degrade on failure, and an exposure to severe outages. An operations team can't limit blast radius without digital service changes prioritised by a product manager.

Availability restoration:

- ***Slow alert acknowledgement.*** An alert takes up to 20 minutes to be acknowledged. The on-call operations bridge analyst needs to classify the alert, look up the correct application support analyst in the on-call schedule, successfully contact that analyst, and handover the alert for acknowledgement.
- ***Slow incident resolution.*** An incident takes hours or days to be resolved:
  - An outsourced application support team can be slow to pick up an L2 incident, if their contractual response time is hours or days.
  - An application support team can be slow to pick up an L2 incident, if they are dealing with planned work or another incident.
  - An application support analyst can bounce an L2 incident back to the operations bridge team, if they need more details or don't believe they are the best-placed responder.
  - An application support analyst can be unable to implement a workaround for an L2 incident, if they lack the permissions to modify service configuration.
  - A delivery team can no longer exist to pick up an L3 incident, if they were disbanded and moved onto new project work after their service was launched.
  - A delivery team can be slow to pick up an L3 incident in office hours, if they are already dealing with planned product features for a deadline.
  - A delivery team developer can be slow to pick up an L3 incident out of hours, if their on-call is best efforts and there is no compensation for on-call inconveniences.
  - A delivery team developer can bounce an L3 incident back to the application support team, if they need more details or don't believe they are the best-placed responder.
  - A delivery team developer can be slow to understand an L3 incident, if they lack prior knowledge of abnormal operating conditions for their digital services.
  - A delivery team developer and application support analyst can find it hard to collaborate on an L3 incident, if their ways of working and tools are different.
  - A delivery team developer can be unable to access telemetry data for an L3 incident, if they lack the permissions to use live telemetry tools.
  - An incident manager, DBA, or network admin can be slow to pick up an L2 or L3 incident, if they are dealing with planned work or another incident.
- ***High volume of L2 callouts***. Incidents can require more L2 escalations than expected. Failure modes are rarely straightforward or repeatable. They may require diagnosis by the application support team, to confirm if a resolution by the operations bridge team is possible. The operations bridge team can only have scripts and runbooks for anticipated failure scenarios.

We've seen these reliability drawbacks add up to substantial financial losses incurred during production incidents, particularly during key trading periods. The most common countermeasures are:

- *L1 delivery team for key periods*. **Delivery teams can provide hypercare and peak support** during service launch and/or busy trading periods, to flatten the time to resolve any incidents. This incurs additional run costs if the delivery teams are compensated, plus opportunity costs due to the lack of feature development.
- *L1.5 application support team*. Digital service alerts can be routed to the application support team as well as the operations bridge team. This speeds up alert acknowledgement and incident resolution. It inevitably increases the workload on the application support team, and calls into question the purpose of the operations bridge team for digital services.

Reactive culture

These drawbacks are specific to root cause analysis, as it is the most common means of collaborative learning in Ops Run It.

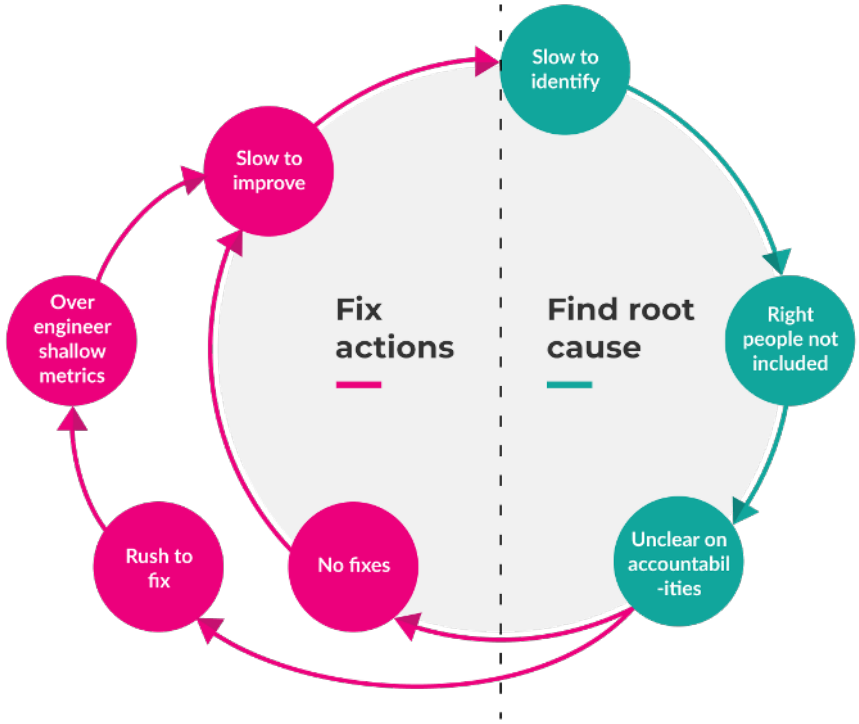
- *Low quality insights.* Root cause analysis sessions produce a few insights of limited value.
  - Attendees lack sufficient time pre-session and in-session to consider events and their consequences during an incident.
  - Attendees are implicitly steered towards identifying a single human or technical error as the sole cause of failure.
  - Attendees can be publicly blamed for causing an incident, if human error is believed to be the sole cause of failure.
  - Attendees can withhold information about their participation in incident response, if they fear blame attribution and ramifications for acknowledging mistakes.
- *Poor insight dissemination.* Problem managers lack the incentives and/or organisational contacts to share a root cause analysis across an organisation. Delivery teams may be entirely unaware when their digital services are involved in production incidents. Knowledge is trapped in silos.
- *Slow time to improve.* A fix action can take days, weeks, or months to be completed. We've seen fix actions stuck in support backlogs for long periods. We've even seen the same action appear multiple times in the same backlog, because multiple incidents for the same digital service have yielded the same unimplemented action.
  - An application support team can be slow to pick up an action, if they are dealing with planned work or another incident.
  - An application support analyst can bounce an action back to the problem manager, if they need more details or don't believe they are the best-placed responder.
  - A delivery team can no longer exist to implement an action, if they were disbanded and moved onto new project work after their service was launched.
  - A delivery team can be slow to pick up an action, if they are already dealing with planned product features for a deadline.
  - A delivery team developer can bounce an action back to the problem manager, if they need more details or don't believe they are the best-placed responder.

Our customers are often aware of the high coordination costs of root cause analysis sessions, and long lead times for fix actions. We usually see these countermeasures:

- *Root cause analysis for major incidents only.* Root cause analysis sessions are only held if an incident is deemed to be high priority. This limits learning opportunities for operations and delivery teams, and ensures weak signals of latent faults within live digital services go unseen.
- *Fix delivery team.* This is sometimes known as a small works team. A fix delivery team implements code fixes for backlog actions, based on analysis from the application support team. This is no guarantee of faster lead times, as the fix delivery team does not own any digital services itself. The affected delivery teams can take days or weeks to accept proposed code changes, and incorporate them into their own deliverables.

We believe these drawbacks are tied to the Ops Run It predisposition of rush to fix. When a production incident happens, identifying fix actions is prioritised over in-depth investigation. Operations and delivery teams have a fixed mindset in which flaws and mistakes are shameful, and concealed from others. This creates a vicious circle, in which root cause analysis sessions are completed as quickly as possible, nobody has the time to acquire meaningful knowledge about their digital services, and the production incidents continue.

Ops Run It - reactive culture



Spotlight

Recognising the problem

A large telco customer I worked with was having trouble with its monolithic systems. They had 100% outsourced delivery and operations teams.

- They were unable to achieve the deployment throughput they needed to meet demand. The operations team managed overnight releases, with a delivery team available to help solve release issues.

- They often had reliability problems after large deployments and peak trading events. Incidents were managed by an operations team with 'pass the buck' handovers to delivery teams, and their Mean Time To Repair (MTTR) was six hours. Delivery teams were asked to do L3 unpaid support, and this led to resentment. In addition, the commercial model for incident response meant it wasn't financially viable to pay the outsourced operations team to fix incidents that weren't P1 or P2.

They had a strong predisposition towards rush to fix. Many fix actions remained on a problem backlog without ever being prioritised or fixed.

This customer understood that they needed to fundamentally change their operating model, and they were interested in You Build It You Run It.

They started on their journey by automating their manual regression testing and monolith deployments, so they could incrementally improve service reliability.

The next step was to look at their overall architecture, and review if the monolith architecture and team structures were the fit for Continuous Delivery and You Build It You Run It. They soon began a journey to rip and replace their monolithic systems, and revisit the interactions between delivery and operations teams at each stage of the process.

Bethan Timmins
Managing Director

Equal Experts
Australia & New Zealand

Spotlight

The special character and the million pound outage

I worked with a large ecommerce customer, where six delivery teams built a monolithic website on top of a vendor commerce platform. The website was run by an operations bridge team and an application support team. The customer wanted to scale up to 25 delivery teams, and replace the monolith with microservices.

One day, we lost the ecommerce website from 0800 to 1300, and it cost us $1.5 million in trade. I was in the incident room, and we battled through a succession of latent faults including poor error handling, inadequate telemetry, a missing cache health check, and invalid character rendering. We slowly realised a single special character typed into a content asset had caused a huge chain reaction, and we'd lost the website as a result.

In the root cause analysis session afterwards, the operations manager declared there were many contributing factors to learn from, not a single human error. The application support team manager flagged his concerns about centralised production support, including knowledge transfer challenges and the plans for 25 delivery teams.

The incident was a turning point. The operations manager was supportive of You Build It You Run It, and delivery teams building operability into microservices.

Together, we gradually experimented with a new operating model in which delivery teams did their own deployments, and went on-call out of hours for critical services. This contributed to a sea change in deployment frequency and incident response times, and it held firm as the number of delivery teams reached 25 and beyond.

Steve Smith
Principal Consultant

Equal Experts
UK

You Build It You Run It

What is You Build It You Run It

An ever-increasing number of organisations have an IT department that includes on-call product teams responsible for building and running digital services. Each team builds and runs long-lived products, and provides 24/7 support for them. This is known as You Build It You Run It, a phrase first coined by Werner Vogels in 2006:

> *Giving developers operational responsibilities has greatly enhanced the quality of the services, both from a customer and a technology point of view. The traditional model is that you take your software to the wall that separates development and operations, and throw it over and then forget about it. Not at Amazon. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service.*
>
> ***Werner Vogels**, Chief Technology Officer at Amazon*

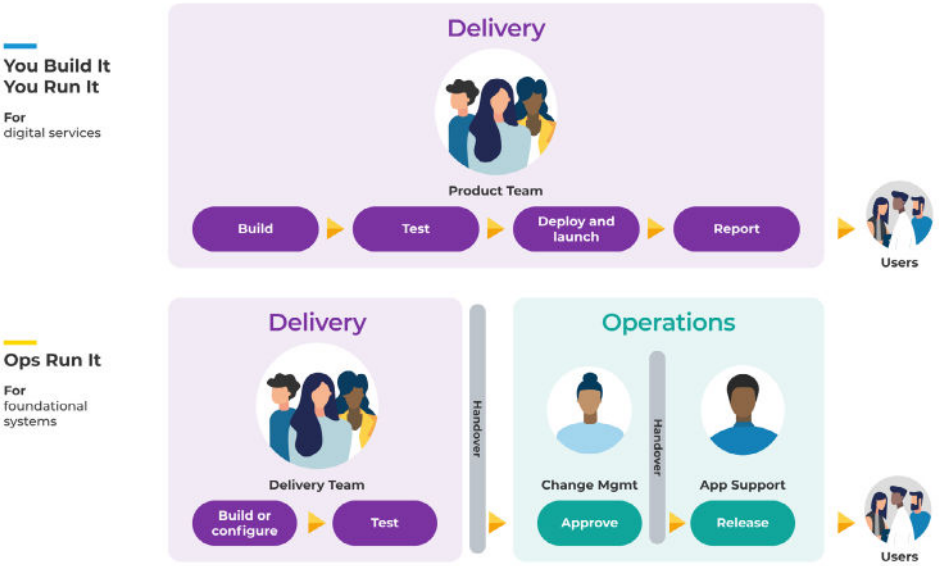
As explained by Jez Humble *et al* in ***Lean Enterprise***, it's a myth that IT governance standards require segregation of duties. You Build It You Run It is compatible with IT Service Management (ITSM), IT Infrastructure Library (ITIL) v3 Service Transition, ITIL v3 Service Operations, **PCI-DSS** for credit card payments, and **SOX** for USA federal securities.

There's no divide between Delivery and Operations functions in You Build It You Run It. There are no short-lived delivery teams. There are long-lived on-call product teams, who own their digital services from inception to live traffic and potentially through to decommission. They are incentivised to move fast, and provide reliability.

Operations teams are likely to exist alongside You Build It You Run It. They will manage self-hosted foundational systems, such as COTS applications and custom integrations. They also remain responsible for lower-level operational capabilities, such as on-premise Domain Name System (DNS) and firewall management.

Every organisation is different, and every You Build It You Run It implementation is different. See our **operating model examples** from our customers.

Deployment throughput in You Build It You Run It



An on-call product team performs a production deployment when they have successfully completed product demos, automated functional testing, exploratory testing, security testing, and any other assurance checks.

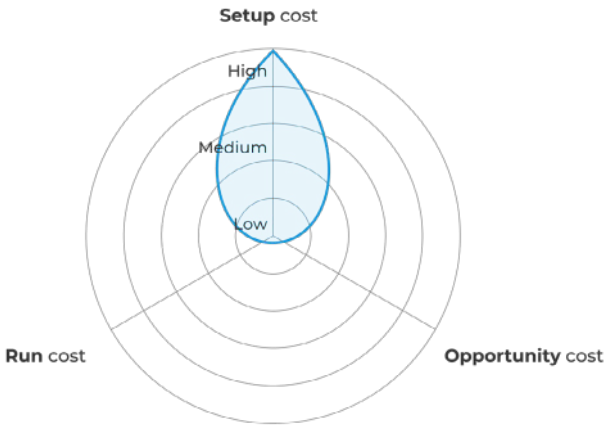
1. *On-call product team reviews change.* If the deployment is a regular, low risk change then the product team chooses a pre-approved change request. If the deployment is irregular or high risk, they add a ticket into the change management queue, and fall back on a CAB meeting with the change management team. In ITIL v3, this is the difference between a standard change and a normal change.

2. *On-call product team performs deployment.* The product team uses a tool such as Jenkins or GitLab CI to orchestrate the deployment.On-call product team performs post-deployment validation. The product team does some post-deployment smoke testing, and continues to monitor live traffic.

3. *On-call product team sends deployment notification.* The deployment process ends with the automated addition of a ticket into the change management queue, notifying the change management team of the deployment and its result.

Deployment throughput costs in You Build It You Run It

Deployment costs are incurred as capex, as they are performed by on-call product teams themselves.



COST TYPE	FREQUENCY	DESCRIPTION	IMPACT	TCO % ESTIMATE
Setup cost	One-off	Launch costs incurred in • Deployment pipeline setup • Change management team time for setup	Capex cost	High
Opportunity cost	Per feature	Can be measured as the cost of delay between product feature readiness (approval) and its launch. Potential revenue lost, missed customer opportunities due to: • Delay waiting for a CAB meeting (irregular, high risk deployments only)	Lost revenue	Low
Run cost	Per deployment	Deployment costs incurred in completing the deployment • Time for performing deployments, and rolling back failed deployments minimised due to no handovers between teams and automation in place.	Capex cost	Low

Service reliability in You Build It You Run It

You Build It You Run It

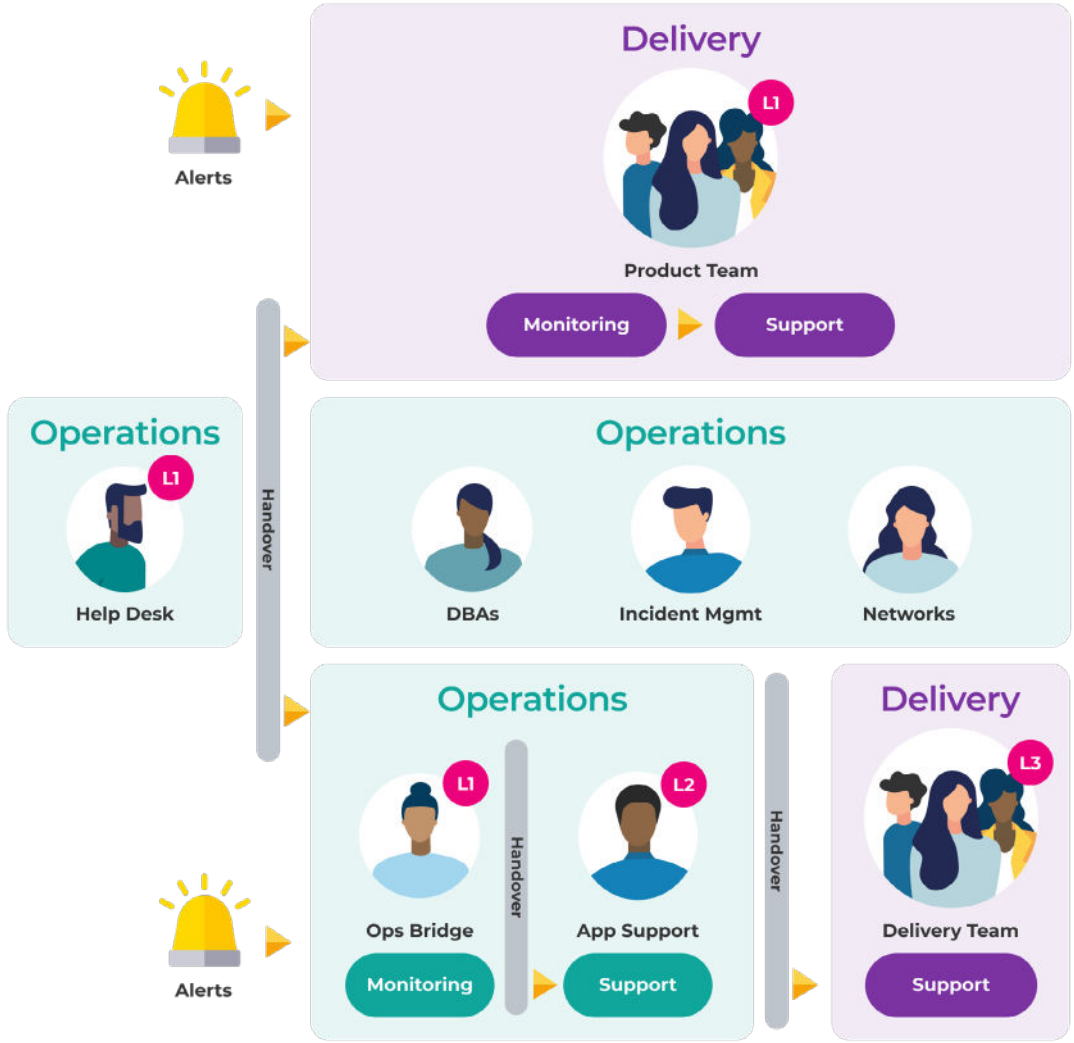
For
digital services

Operational Enablers

For
everything

Ops Run It

For
foundational systems



An on-call product team offers 24/7 production support, and they can modify all aspects of a digital service:

- Alert definitions
- Code
- Configuration
- Data
- Deployments
- Infrastructure definitions
- Logging
- Monitoring dashboards

You Build It You Run It co-exists in a hybrid operating model with **Ops Run It**, and it's important to have specialist operational teams as well as generalist, cross-functional product teams. The same operational enablers offer their scarce, deep expertise to on-call product teams for digital services, and to the application support team for foundational systems. For example, a shared DBA team can assist with database provisioning, performance, and operations.

For governance, the senior manager for the on-call product teams is accountable for all aspects of digital service reliability. The product teams are responsible for day-to-day work in availability protection and availability restoration. They track their costs on a team-by-team basis, and make them visible for senior leaders.

Availability protection in You Build It You Run It

An on-call product team is responsible for availability protection. This means proactively monitoring service telemetry and updating digital services during working hours. A product team observes service health checks, logs, and metrics plumbed into different dashboards, in telemetry tools such as **AWS CloudWatch** or **Grafana**. They can alter the monitorable events whenever necessary, to improve the information value of their telemetry data.

A product team also updates their digital services by adding infrastructure capacity, making configuration changes, applying fixes, and updating alerts as necessary. This is prioritised alongside feature development, as it is the product team themselves who are on-call.

Availability restoration in You Build It You Run It

An on-call product team is responsible for availability restoration. This means reactively responding to production alerts in and out of working hours, including evenings, weekends, and bank holidays. All service alerts are expected to be resolved by the on-call product team, and incident response is consistently prioritised over feature development.

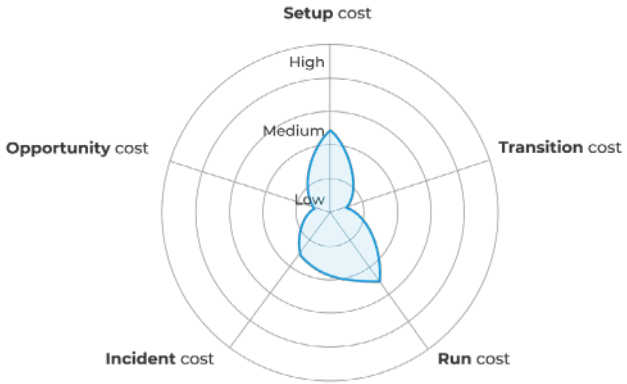
When an availability target is breached, an on-call product team developer receives an automated alert, via an incident response platform such as **PagerDuty** or **VictorOps**. The responder acknowledges the alert, and a ticket is automatically created in a ticketing system such as ServiceNow, by the incident response platform. The responder classifies and prioritises the incident, and adds more team members to the incident as required.

Responders observe their real-time service telemetry data, to understand the drift from normal to abnormal operating conditions. They diagnose the incident via their heuristics, innate service knowledge, and telemetry data. They attempt to restore availability via additional infrastructure, code changes, data fixes, configuration changes, rollbacks, and telemetry changes.

For a high priority incident, the entire team may swarm on incident response, to minimise service unavailability. One team member acts as incident commander, to coordinate with other product teams involved in incident response, and to manage communications with senior stakeholders. Alternatively, You Build It You Run It is 100% compatible with ITIL v3, and an incident manager could be invited by the team to act as incident commander for the duration of an incident.

Service reliability costs in You Build It You Run It

If an IT department has segregated Delivery and Operations functions, on-call funding can be either capex or opex. You Build It You Run It is most effective when **Delivery owns the on-call budget**, with the senior manager accountable for on-call funding as capex spend. However, a frequent pitfall in large organisations is **on-call budget stuck in Operations**, leaving the senior Operations manager accountable for on-call funding as opex spend.



COST TYPE	FREQUENCY	DESCRIPTION	IMPACT	TCO % ESTIMATE
Setup cost	One-off	Launch costs incurred in • License purchases • Product team time for telemetry install • Product team time for schedule agreement • Product team time to setup live access • Product team time for any operational training necessary	Capex cost	Medium
Transition cost	One-off	Launch costs incurred in • Product team time for runbooks	Capex cost	Low
Run cost	Ongoing	Regular costs incurred in application support team time for • Deploying code changes • Applying data fixes • Making configuration changes • Adding infrastructure capacity • Monitoring operating conditions • Performing rollbacks • Updating telemetry tools • Doing on-call standby out of hours	Capex cost	Medium
Incident cost	Per incident	Incident response costs incurred in application support team time for • Investigating and diagnosing problems • Identifying and agreeing on solutions including code changes, configuration updates, adding infrastructure capacity • On-call callout out of hours	Capex cost	Low to medium
Opportunity cost	Per incident	Can be measured as the cost of delay between incident start and finish. Caused by service unavailability, missed opportunities with customers, and delays in further feature development	Revenue loss and costs incurred	Low

Learning culture in You Build It You Run It

We describe the You Build It You Run It learning culture as predisposed to act on insights. When there is an incident of consequence, the priority for an on-call product team is to acquire a deep understanding of the problem and its resolution, and then broadcast the accumulated knowledge across the organisation.

Generate insights

After an incident, an on-call product team is immediately allocated a period of incident analysis, for each team member that participated in incident response.

It includes interviews, establishing key events during the incident, and uncovering any common learning themes. This is scheduled by the product manager, or a problem manager in accordance with ITIL v3.

Afterwards, there is a collective post-incident review. The purpose is to generate as many insights as possible, and turn them into improvement actions where necessary. The session format is:

- Review the incident analysis reports from different team members.
- Build a shared understanding of the incident, according to the organisational context.
- Explore the different perspectives of the team members involved in incident response.
- Discuss the human and technical contributing factors to the incident.
- Identify the most valuable learnings from the incident, and any associated improvement actions.
- Blend the newly acquired knowledge into a compelling narrative.

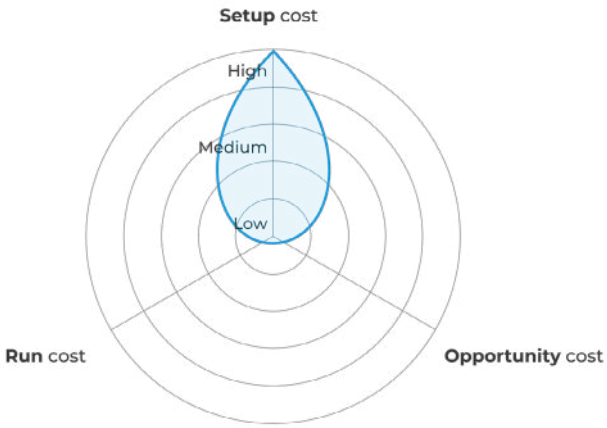
A team member publishes the incident narrative to the entire organisation, and there may also be internal presentations and walkthroughs.

Implement improvement actions

After the post-incident review, a team member records any improvement actions in the product backlog, using a ticketing system such as Jira or Trello. This includes human factors such as accountability changes, as well as technical factors such as code fixes and configuration changes. As usual in You Build It You Run It, the product manager is responsible for balancing the priorities of those actions versus planned product features.

Learning culture costs in You Build It You Run It

Culture costs are capex, as they are incurred by on-call product teams themselves.



COST TYPE	FREQUENCY	DESCRIPTION	IMPACT	TCO % ESTIMATE
Setup cost	One-off	Launch costs incurred in • Product team time for incident analysis training • Product team time for incident review training • Product team time to setup cross-team playbacks, walkthroughs, knowledge sharing	Capex cost	High
Opportunity cost	Ongoing	Can be measured as the cost of delay between an improvement as an idea and in practice. Potential revenue lost, costs incurred, missed customer opportunities, due to delay in improvements	Lost revenue Capex cost	Low
Run cost	Ongoing	Costs incurred in putting an improvement idea into practice	Capex cost	Low

Spotlight

Moving to You Build It You Run It on O2 Priority

I joined O2 in 2016, one of the UK's major telecommunication companies, at a time when it wanted to revamp its digital services. O2 Priority was a customer loyalty service, which was vital to its customer retention strategy. At the time, O2 Priority was developed by a delivery team, and supported by an operations team. There were plenty of problems:

- The service was plagued with many production incidents
- Deployments were fragile, and often had to be rolled back
- A painful deployment experience limited the delivery team's ability to deliver new features.
- Every change request slowly went through detailed scrutiny
- Deployments were done out of hours by the Ops team
- The Ops team had other responsibilities, and could not fully monitor the service or automate the deployment pipelines

A cross functional team was put together to re-architect and rebuild the components of O2 Priority, its infrastructure and pipelines. The team implemented You Build It You Run It, and owned delivery and operations of the service.

Fully automated pipelines were put into all environments, to ensure Continuous Delivery of both microservices and infrastructure. Zero-downtime deployments gave O2 sufficient confidence that creating change requests and attending CAB meetings were no longer required unless downtime was unavoidable. Enhanced monitoring capabilities meant the team could be proactive in detecting and fixing operational issues, faster than ever before. Measures such as circuit-breakers were also put in place, to gracefully manage third party failures downstream which were capable of affecting the service.

The outcome of this change in operating model was much shorter lead times, a stable O2 Priority service with no major customer impacting incidents, and happier customers.

Learn more about our partnership with O2 here.

Ogonna Iwunze
Operability Engineer

Equal Experts
UK

Benefits of You Build It You Run It

You Build It You Run It benefits come from team empowerment, and clear operability incentives. There's no hard divide between siloed delivery and operations teams. There are no time-consuming handoffs, nor any diffusion of responsibility. A product manager is incentivised to constantly balance the prioritisation of operational features alongside product features, because they're accountable for service reliability. A product team is incentivised to build operability into their digital services, because they're on-call out of hours for their own work.

Deployment throughput

- *Fast change approvals.* Frequent, low risk deployments have pre-approved standard change requests, which involve no change management team time. Infrequent, high risk deployments may still require a CAB meeting, and incur similar change approval delays as Ops Run It.
- *Fast deployments.* Deployments take minutes to complete successfully. An on-call product team can perform a deployment in working hours with no handoffs, and they can fully automate their own deployment pipeline to minimise deployment failures.
- *Focus on outcomes.* Digital services are created to deliver customer outcomes. Live traffic data is used to create product hypotheses, which are tested against customer feedback.
- *Low knowledge synchronisation costs.* An on-call product team can store most of its service documentation within the code itself, and use its own runbooks to store acquired operational knowledge. Knowledge synchronisation takes minutes between team members, as they already work within the digital service on a daily basis.
- *No scheduling conflicts*. An on-call product team can reschedule its deployments as and when necessary, without any conflicting workload priorities.
- *Low run costs.* The costs of performing deployments and rolling back failed deployments can be extremely low. An on-call product team has sole responsibility for their automated deployment pipeline, and can devote as much time to it as necessary.
- *Proactive change management.* Change managers can become involved in digital service delivery before and during live traffic. They can contribute to deployment patterns, ask for changes to post-deployment health checks, and receive a fully automated audit trail from a deployment pipeline.

These benefits add up to low opportunity costs for a digital service. You Build It You Run It minimises the cost of delay between product feature readiness and launch, because change approvals and deployments can happen so quickly. This also applies for additional infrastructure capacity, library upgrades, and security patches.

Service reliability

We usually see an on-call product team with cross-functional collaboration, cross-pollination of skills and experiences, a shared sense of purpose, and a high level of job satisfaction.

Availability protection:

- *Informative telemetry*. Alerts, logs, metrics, and dashboards are continually refined, in order to increase their information value and describe operating conditions. Telemetry events can be modified and deployed by the on-call product team immediately.
- *Sufficient documentation.* An on-call product team is highly motivated to create and maintain up-to-date architecture diagrams, alert threshold guides, configuration settings, and runbooks to operate their own digital services.
- *On-call satisfaction.* Doing L1 support means on-call product team members can be compensated fairly for the inconvenience of out of hours support, and the disruption to their lives outside of work.
- *Adaptive architecture.* Liability for out of hours callouts encourages an on-call product team to design digital services that can gracefully degrade on failure, with a small blast radius and minimal exposure to cascading downstream failures.
- *Low transition costs*. An on-call product team has its digital services in a state of continual service transition. Operational features are continually added, and checks of operational readiness can be automated.

Availability restoration:

- *Fast alert acknowledgement.* Alerts take seconds or a few minutes to be acknowledged, because they are immediately routed to an on-call product team member without any handoffs.
- *Fast incident resolution*. An incident takes minutes or hours to be resolved, as alerts are routed to a long-lived product team who intimately understand the failing digital service, have full access to live telemetry data, and are able to make immediate code changes if necessary.
- *Low number of callouts.* An on-call product team liable for out of hours callouts is incentivised to fix intermittent alerts and low priority latent faults whenever they occur, and they have the ability to modify telemetry events and alert definitions as necessary.
- *Low to medium incident costs.* An on-call product team able to add infrastructure capacity, deploy code changes, and roll back failed deployments whenever necessary is well placed to minimise incident costs and callout charges.

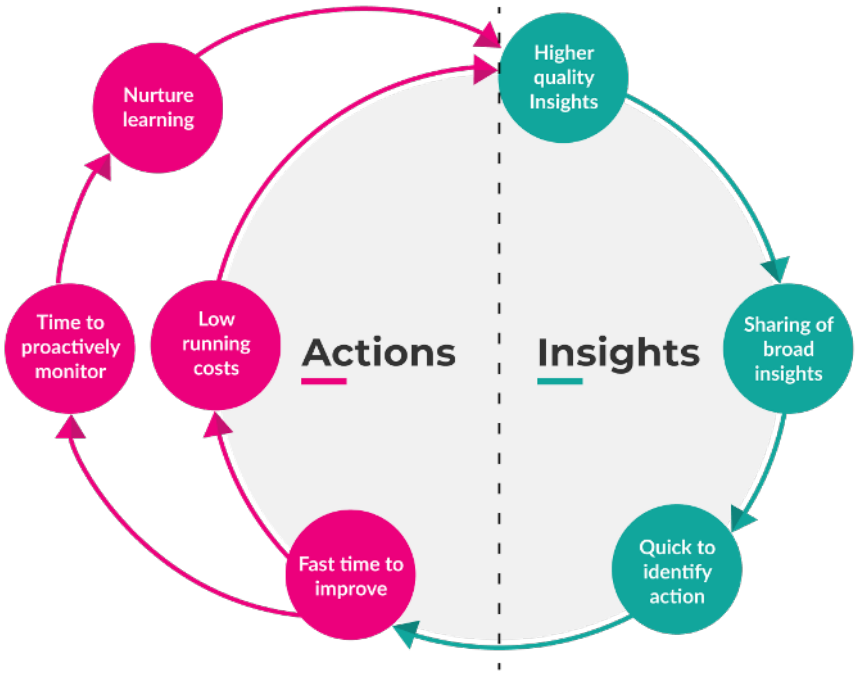
These restoration benefits produce a low opportunity cost per incident. Potential revenue loss and operational costs are significantly reduced by an on-call product team that consistently resolves a majority of incidents in minutes, rather than hours.

Learning culture

- *High quality insights.* Splitting the post-incident process into independent investigations, group investigation, and action planning creates powerful insights into the human and technical factors, knowledge gaps, and operational surprises related to an incident. The diverse perspectives of different product team members provide valuable context, and nobody is blamed for their involvement in an incident.
- *Broad insight dissemination.* On-call product teams are incentivised to collaborate on sharing incident findings between teams, so current and future team members can benefit from the newly acquired knowledge.
- *Fast time to improve.* An improvement action takes hours or days to be implemented. Actions are added to the product backlog, and prioritised by the product manager like all planned work. On-call product teams have every reason to improve feature delivery and customer experience as quickly as possible, as they are liable for further out of hours callouts. This includes both high and low priority actions.
- *Nurture a learning organisation.* Encouraging a growth mindset allows a learning organisation to slowly develop. Leaders who understand what it means to create a learning organisation are in a strong position to foster strong collaboration in turbulent markets and future crises.
- *Low run costs.* The time taken to implement improvement actions is low. It's possible for a good idea in a post-incident review to be implemented that very same day, without any time-consuming handoffs to other teams.

We see these benefits as outcomes from the You Build It You Run It predisposition of acting on insights. On-call product teams have a growth mindset, in which flaws and mistakes are learning opportunities to be publicly shared across the organisation. This creates a virtuous circle.

You Build It You Run It - learning culture



Hutchison 3G

Spotlight

You Build It You Run It at Hutchison 3G

At Hutchinson 3G (Three Mobile), Equal Experts was involved in building and running a customer loyalty platform, where customers can claim discounts based on loyalty points.

Our development team built an iOS app with Swift, an Android app with Kotlin, and a middleware API with Spring Boot and Kotlin on AWS. We used You Build It You Run It, self-servicing deployments and doing out of hours on-call.

Our deployment frequently moved from monthly to daily. We used feature flags, which was a big boon to product owners as small features were rolled out incrementally and features could be prioritised based on live customer data.

Because we were the people on-call, we had a real emphasis on traceability. Business metrics and diagnostics were built around expert domain knowledge gained during development. Every request was logged, aggregated, and visualised in real-time on big screens so stakeholders could see live customer behaviours. Faults were detected at an early stage, support tickets were minimised and quickly resolved, and insights from production support helped us to better design the apps.

The backend coupon management platform was a COTS application, managed by an outsourced supplier.

Their central operations team looked after multiple coupon customers, and they had slower response times than us for major incidents. Hutchinson 3G had a central operations team to monitor the overall system, and the difference in response times was noticeable. They had full access to our monitoring dashboards, which gave them even more confidence in You Build It You Run It.

Harish Kannaro
Lead Developer

Equal Experts
UK

Drawbacks of You Build It You Run It

Deployment throughput

- *High setup cost.* It takes weeks or months to establish a new, streamlined change management process for digital services.
  - Moving accountability for change approvals from a change management team to product managers takes weeks or months.
  - Moving accountability for production deployments from an application support team to on-call product teams takes weeks or months.
  - Creating a new change approval process in a central ticketing system like ServiceNow takes weeks or months.

The countermeasure here is for product teams to solve problems for the change management team, not just for themselves. This can include providing change managers with:

- A list of change approval criteria that product managers will consistently follow.
- A set of automated integrations for change requests into the central workflow system.
- A standing invite to all product planning meetings and pre-deployment product demos.
- An automated email notification whenever a deployment has happened.
- An automated web page showing recent change requests and the associated deployments.

Service reliability

- *High setup cost.* It takes weeks or months to establish incident management for digital services.
  - Moving accountability for incident management from an application support team to on-call product teams takes weeks or months.
  - Creating a new incident management process in a central workflow system like FreshService takes weeks or months.
  - Procuring licenses for an incident response platform such as PagerDuty takes weeks, and can become expensive at scale.
  - Enabling production access rights for telemetry tools, deployment servers, and incident management for on-call product teams takes weeks.
  - Establishing out of hours schedules and operational training for on-call product teams takes weeks.
- *Risk of high run cost.* Remuneration for multiple product team developers on call for multiple digital services can be higher than a single application support team analyst on call for all digital services.

In our experience, organisations underestimate the high setup cost and overestimate the risk of a high run cost with You Build It You Run It. The setup cost can be reduced by collaborating with your incident management team on:

- An incident management process that on-call product teams will consistently follow.
- A set of automated integrations for incidents into the central workflow system.
- A standing invite to all product planning meetings and Chaos Days.
- A per-team license setup for telemetry tools and incident response tools, where possible.

Learning culture

- ***High setup cost.*** It takes weeks or months to create a new incident review process for digital services.
  - Moving accountability for problem management from an application support team to on-call product teams takes weeks.
  - Setting up cross-team knowledge sharing after post-incident reviews takes weeks.
  - Incident analysis training for on-call product teams takes weeks or months.
  - Establishing psychological safety within on-call product teams for post-incident reviews takes weeks or months.
  - Creating a blameless culture between on-call product teams takes weeks or months.

There aren't any obvious countermeasures here. An upfront investment in effective incident analysis training is vital, as moving away from root cause analysis is a subtle paradigm shift. It's an entirely different way of understanding organisational problems, and it takes time to build up momentum for continuous improvement.

Spotlight

You Build It You Run It skepticism in financial services

I worked with a large financial services company that wanted to use modern techniques and tools to revamp its digital offering. The existing digital product was run on-prem using Ops Run It, and it had plenty of problems.

For example, when a public-facing service was slow to respond to customer interactions, the development team was asked why, and their response was “we’ll ask the Ops team”. They had no knowledge of how the product performed in the customers’ hands, and didn’t have the diagnostics or metrics to answer such questions.

We spun up a couple of teams, to build infrastructure and bootstrap new digital capabilities. We brought forward new ideas that represented a 180 degree turn for the company. These included using cloud infrastructure, building a digital platform, moving to Continuous Delivery, and adopting You Build It You Run It. Each of them took months to be signed off.

Things slowly took a turn for the better. Cloud became a first-class citizen in the fabric of the organisation. Delivery teams were given operational control of the services they built, and our platform team gave them early access to observability and operational tooling.

As a result, we saw an increase in deployment frequency, and operational engagement from the developers.

Although the company made a lot of progress, there were still plenty of caveats. The need for regular penetration exercises delayed deployments, and a preference for out of hours deployments was hard to shake off. Still, I could see how You Build It You Run It had started to shape a financial services company into a software delivery powerhouse.

Charles Tumwebaze
Lead Developer

Equal Experts
South Africa

Principles

We recommend the adoption of these operating model principles before you start to implement You Build It You Run It for your digital services. All of our You Build It You Run It practices tie back to these principles, which represent our thinking on how operating models succeed.

Operating models are insurance for business outcomes

An operating model is an insurance policy. It protects the business outcomes you're trying to achieve, in exchange for a premium.

Different insurance policies offer different levels of protection, at different premiums. The more valuable the contents of your home, the higher the premium you pay for the right house insurance.

Different operating models offer different levels of protection, at different premiums. The more valuable your business outcomes, the more the risk of loss, the more of a premium you pay for the right operating model.

Business outcome protection is a mix of revenue cover and cost avoidance. The weighting of the mix varies between organisations. A private sector organisation with an ecommerce website might insure for 80% revenue cover, 20% cost avoidance. A public sector organisation with a non-transactional website might want 5% revenue cover, and 95% cost avoidance.

Operating models are selected on financial exposure and product feature demand

An operating model is designed to safeguard levels of financial exposure, and satisfy levels of product feature demand for business outcomes.

You need to understand the financial exposure and product feature demand for each of your software services, and select the most appropriate operating model on a case by case basis. The same blanket operating model for all software services cannot offer cost effective insurance for multiple levels of financial exposure and product feature demand.

Business outcomes can be expressed as opportunity costs, in terms of deployment throughput and service reliability. If they map onto significant demand for new product features, the deployment target needs to be weekly deployments or more. If they represent a substantial amount of revenue and/or costs at risk upon unavailability, the availability target needs to be 99.9% or higher. At that point, a higher premium is paid in implementation efforts, in order to minimise the opportunity costs linked to the business outcomes.

Operating models are most effective when rapid restoration comes first

An operating model divides service reliability into availability protection and availability restoration. Business outcome protection is far more effective when rapid availability restoration is prioritised over perfect availability protection.

Your production environment is a complex group of different systems, services, and processes, with unpredictable interactions in unrepeatable conditions. Some failure classes are preventable, others are not, and your production environment is always vulnerable to failure. We advise our customers that being able to rapidly restore availability is more important than having fewer periods of unavailability.

Favouring a high quality incident response over zero incidents means customers enjoy a smoother experience upon failure, and business outcomes can continue to be met under partial degradation.

Operating models are powered by feedback

An operating model is dependent on fast feedback loops. The tighter the feedback loops, the easier it is for people to collaborate with one another, and achieve the desired customer outcomes.

Your organisation is full of knowledge on deployment throughput, service reliability, customer needs, commercial imperatives, ways of working, production incidents, and much more. Allowing that knowledge to be siloed within different individuals and teams has an insidious impact on organisational performance. Feedback signals on customer satisfaction, product deliverables, and the actions of individuals are slower and weaker. This constantly hampers improvement efforts.

Creating feedback loops within and between teams ensures that newly acquired knowledge can be rapidly disseminated to your entire organisation. Amplifying feedback prevents the same problems from repeating themselves in different digital services, and allows people to perform their work with the cumulative knowledge of your entire organisation. This makes building and running a digital service an easy habit, rather than a laborious chore. See ***Atomic Habits*** by James Clear.

Spotlight

Starting You Build It You Run It too soon

We started to work with a major UK retailer shortly before the COVID-19 pandemic hit the UK. The company had ambitions to modernise its technology estate to offer new services to customers. We built a digital platform to scale delivery up to many teams, and implemented new digital capabilities for customers. We all saw You Build It You Run It as a key part of our technology strategy.

We publicly launched new digital services at a time when uncertainty was at its highest for society. In the early days post-launch, the retailer's high street shops were closed, people weren't commuting into towns, and uptake of the new services was very low. There weren't many opportunities to learn from customers.

As a result, product teams scheduled to go on-call became demotivated. It wasn't easy to experiment and gain insights into customer behaviours, as business was down. Some teams successfully lobbied to delay on-call support until business started to pick up again, which thankfully has happened since then.

In times of low customer uptake, we learned it's really important to plan which experiments are most important for learning, what data is needed for those experiments, and when to pivot.It would have helped us to understand how to differentiate between different levels of on-call support, and therefore the team shapes and sizes needed for You Build It You Run It.

Ann Yong
Delivery Lead

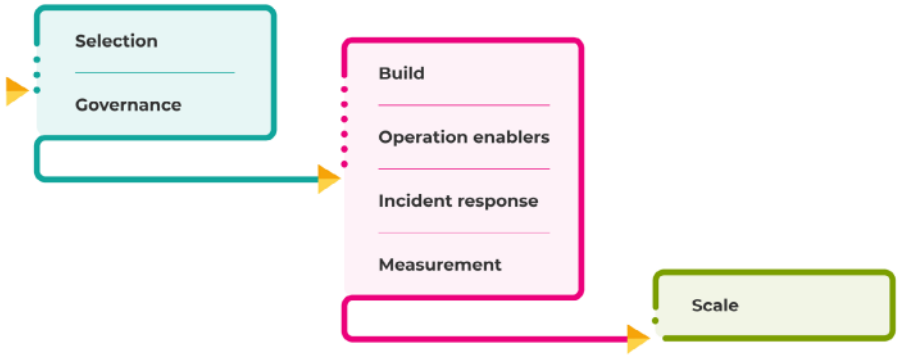
Equal Experts
UK

Practices

The You Build It You Run It practices in this playbook are technology and vendor-agnostic, so you can determine the best adoption pathway based on the context of your organisation. While we're strong proponents of cloud computing, serverless, microservices, and open-source technologies, we've seen You Build It You Run It succeed in very different environments.

Transitioning from Ops Run It to You Build It You Run It is a challenging process. There's no repeatable recipe, because every organisation is different and has its own unique context. We can simply offer a heuristic, which describes a sequencing of You Build It You Run It adoption that we've seen succeed before. It's important that you experiment with these practices, and gradually learn what the right approach is for your organisation.

You Build It You Run It - practices



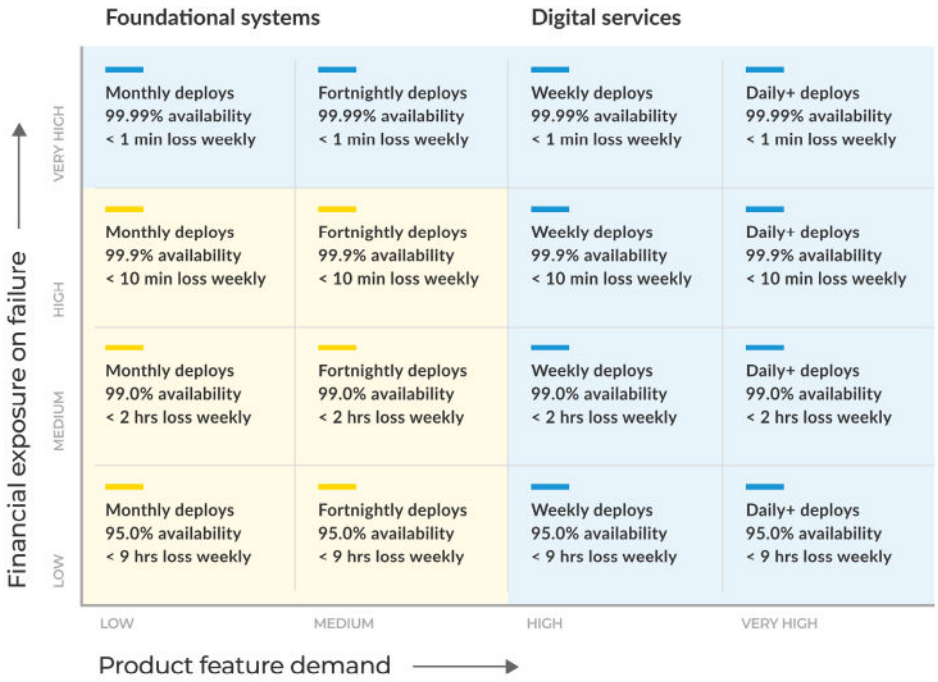
These practices achieve a delicate balance between operability incentives and run costs, such as downgrading availability targets in the months between peak business periods.

Selection

These practices refer to the initial selection of an operating model. We believe a hybrid operating model is essential, because we agree with Martin Fowler's statement that a software service is either a **strategic differentiator or a utility**. We consider a digital service to be a strategic differentiator, and a foundational system to be a utility.

We've created an operating model selector. Once you understand your failure tolerances and different levels of feature demand, you can map out which operating model is right for you.

Operating model selector



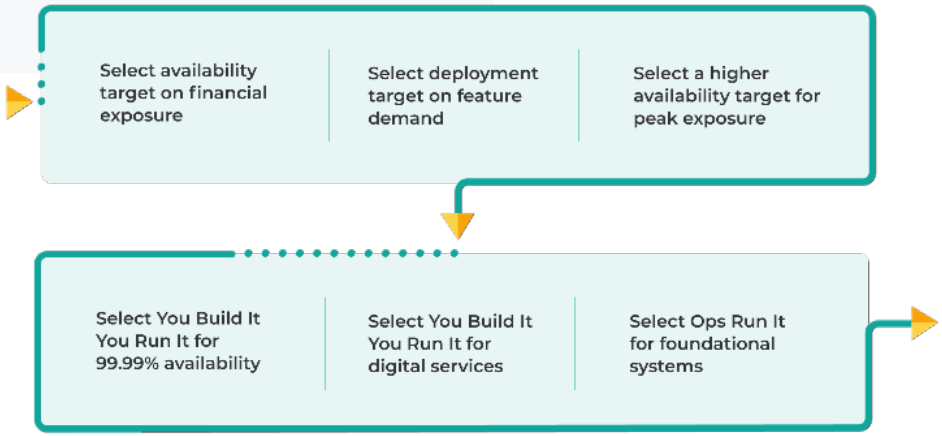
A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

You Build It You Run It

To help understand which operating model is best for your software service, you can follow these selection practices. If you don't use them, you'll likely fall into the **Linear run cost pitfall**.

You Build It You Run It - selection practices



These practices are linked to our principles of **operating models are insurance for business outcomes**, and **operating models are selected on financial exposure and product feature demand**.

Select an availability target on financial exposure

Financial exposure is the maximum revenue loss and costs a software service can incur upon failure. An availability target is a desired level of availability, and expressed as a number of nines. An additional nine of availability means more implementation effort.

Calculate an availability target for a software service via this process:

1. *Estimate financial exposure bands for availability levels.* This is a two-step process for all software services:
    a. Estimate high, medium, and low exposure bands on organisation-wide financial forecasts, and historic incident losses. We ask "what data do we have for how much money we could lose in an hour?"
    b. Assign the availability levels 99.9%, 99.0%, and 95.0% to the high, medium, and low exposure bands. 99.99% can be used if your organisation has a genuine need for extreme reliability. We ask "for this exposure band, how much unavailability are we actually willing to tolerate - and how much of the exposure are we able to accept as a loss?"

2. *Calculate the availability target, by estimating financial exposure.* This is a multi-step process:
    a. A product manager estimates their service exposure based on their own financial forecasting. We ask "what's the maximum revenue loss and costs we'll incur if this software service is unavailable for an hour?"
    b. Automatically link a service exposure to the highest exposure band it fits into. We ask "what's the financial exposure band that covers this service exposure?"
    c. Automatically link a service exposure onto the availability target for a particular exposure band. We ask "what's the availability target for the financial exposure band that covers this service exposure?"

3. *Periodically reflect on financial exposure and availability.* This is for all software services. Compare recent incident losses with our availability target calculator, on at least a quarterly basis. We ask "do we have any new financial data that warrants an update to our financial exposure bands, our availability levels, or the availability targets for our software services?"

Assume a furniture retailer has a self-hosted COTS ecommerce platform, a custom bedroom frontend, and a custom appointments frontend. In step 1, financial losses from prior incidents for multiple software services are examined. Their different traffic profiles and different incident losses allow for an arbitrary grouping into financial exposure bands of $7K, $100K, and $800K loss in one hour.

MAXIMUM FINANCIAL EXPOSURE IN AN HOUR
$7,000
$250,000
$800,000

The furniture retailer examines its appetite for unavailability at its different exposure bands, and commits to 95.0%, 99.0%, and 99.9% as its availability levels. For example, for the $800K exposure band 99.9% of a week is 167 hours 49 mins 55 secs, so the tolerable unavailability in one week of 0.01% is 10 mins 5 secs.

MAXIMUM FINANCIAL EXPOSURE IN AN HOUR	AVAILABILITY LEVEL	TOLERABLE UNAVAILABILITY IN A WEEK
$7,000	95.0%	8:24:00
$250,000	99.0%	1:40:48
$800,000	99.9%	0:10:05

Product managers then use their service forecasts to estimate a maximum financial exposure of $810K in an hour for the ecommerce platform, plus $6K and $200K for appointments and bedroom frontends.

Each software service is matched to an availability target via its service exposure. For example, the bedroom service is assigned to 99.5% availability, as its service exposure of $350K in an hour exceeds the 99.5% exposure band of $100K.

SOFTWARE SERVICE	MAXIMUM FINANCIAL EXPOSURE IN AN HOUR	AVAILABILITY TARGET
ecommerce-platform	$810,000	99.9%
appointments	$6,000	95.0%
bedroom	$200,000	99.0%

The furniture retailer values its financial exposure bands once a quarter. Financial losses from prior incidents are reviewed, and if necessary the financial exposure bands are recalculated. This ensures the risk of financial exposure is revalidated as the business changes.

Indirect financial losses caused by reputational damage also need to be considered. In the private sector this includes customer credits and subscription cancellations. In the public sector, it's employee time spent on manual, paper-based fallbacks. Reputational damage needs to be tied into core business metrics such as customer lifetime value and customer satisfaction, and it can be tracked using **Net Promoter Score**.

Select a higher availability target for peak financial exposure

Calculate a different availability target for a software service during a peak business event, such as Black Friday. This is a repeat of **selecting an availability target on financial exposure**.

This involves a product manager calculating peak and non-peak financial exposures for their software service, based on their financial forecasting. The availability target for the software service is then upgraded just before the peak business event, and downgraded just after the peak business event.

This is an effective cost optimisation, that doesn't affect operability incentives. It replaces the notion of **'eyes on glass' monitoring and peak support in Ops Run It**. It ensures a balance between service reliability and run costs for You Build It You Run It.

Select a deployment target on feature demand

Product feature demand is customer demand for new product features.

A deployment target is a required level of deployment frequency. An increase in frequency means more implementation effort.

Calculate a deployment target using this process:

1. *Estimate feature demand bands for all software services.* We estimate product feature demand based on organisation-wide financial forecasts, customer research, and live analytics. We ask "how much demand is there in a month for product features?"

2. *Match deployment frequency levels to feature demand bands.* We match a feature demand band with a deployment frequency. We ask "for this feature demand band, how often do we need to deploy product features to satisfy demand?"

3. *Estimate feature demand for a software service.* A product manager estimates their feature demand based on customer research and analytics. We ask "what's the feature demand we'll see from customers, for this software service?"

4. *Calculate the feature demand band for a software service.* We automatically link service demand with the feature demand band it fits into. We ask "which of our feature demand bands covers this service demand?"

5. *Calculate the deployment target for a software service.* We automatically link service demand onto the deployment target for a particular feature demand band. We ask "what's the deployment target for the feature demand band that covers this service demand?"

6. *Periodically reflect on feature demand bands and deployment targets.* We compare recent analytics with our deployment target calculator, on at least a quarterly basis. We ask "do we have any new data that suggests an update to our feature demand bands, or our deployment frequency levels?"

The furniture retailer defines relative feature demand bands, from low to very high. The deployment frequency of one software service relative to another is recognised as the most important property. Relative feature bands allow the unique characteristics of different furniture domains to be taken into account.

MAXIMUM FEATURE DEMAND IN A MONTH
Low
Medium
High
Very High

The furniture retailer identifies monthly, fortnightly, weekly, and daily deployments as its deployment frequency levels, and ties them to feature demand bands.

MAXIMUM FEATURE DEMAND IN A MONTH	DEPLOYMENT FREQUENCY
Low	Monthly
Medium	Fortnightly
High	Weekly
Very High	Daily

Product managers estimate low demand for the ecommerce platform, high demand for the appointments frontend, and very high demand for the bedroom frontend.

Each software service is matched to a deployment target by its service demand. For example, the bedroom service is assigned to daily deployments, as its service demand is believed to be very high.

SOFTWARE SERVICE	MAXIMUM FEATURE DEMAND IN A MONTH	DEPLOYMENT TARGET
ecommerce-platform	Low	Monthly
appointments	High	Weekly
bedroom	Very High	Daily

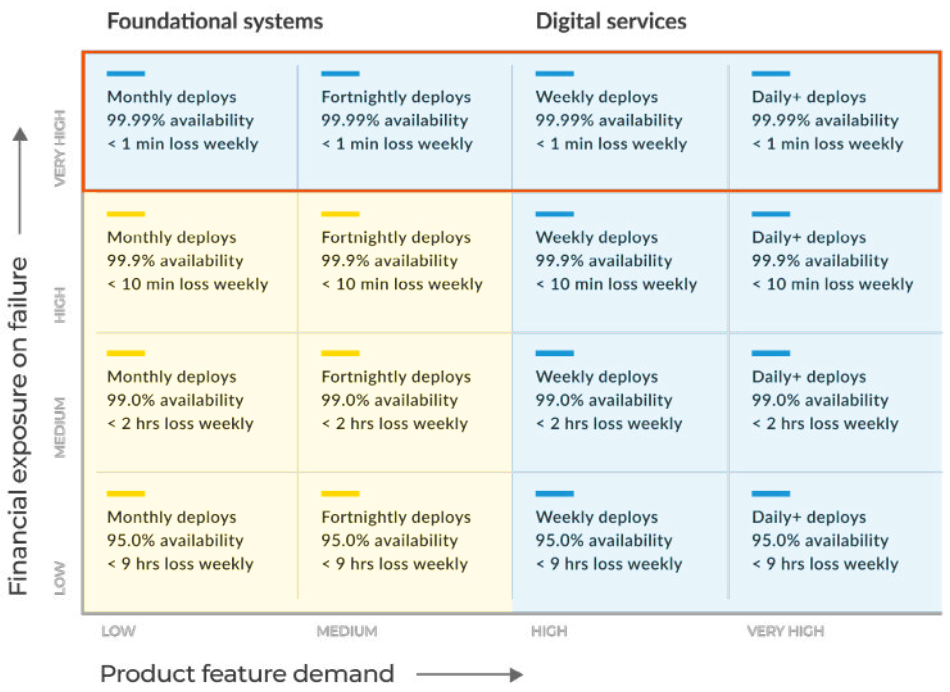
Select You Build It You Run It for 99.99% availability

Adopt You Build It You Run It for any foundational system or digital service, when you require extreme reliability:

- 99.99% availability protection.
- 1 minute of tolerable unavailability per week.

It's rare for our customers to truly require 99.99% availability. The engineering effort involved is enormous.

Operating model selector



A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

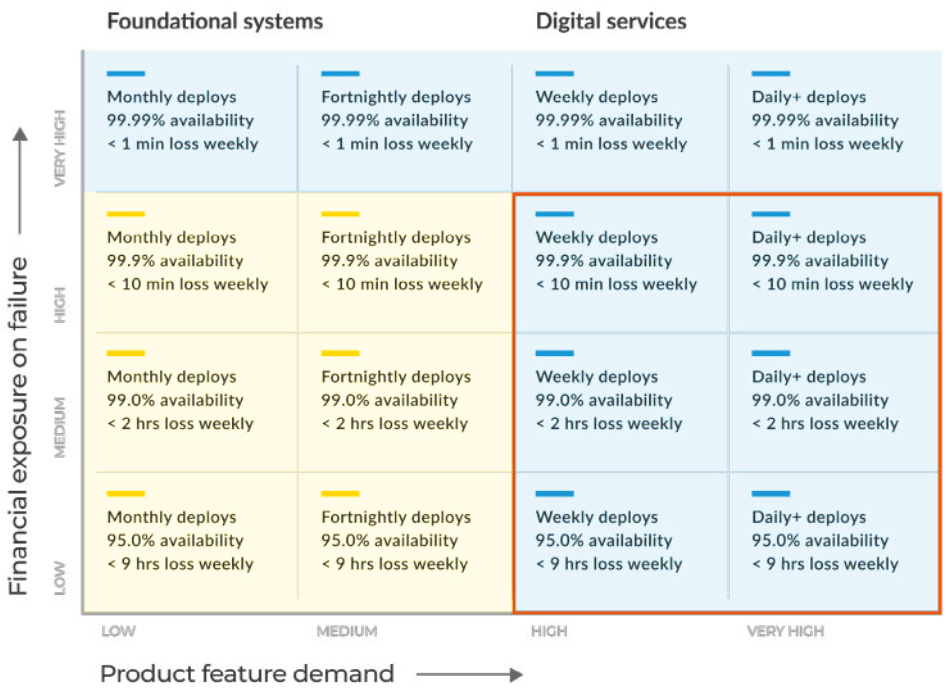
You Build It You Run It

Select You Build It You Run It for digital services

Adopt You Build It You Run It for your digital services, when you have these desired outcomes:

- Weekly, daily, or more frequent deployments.
- 95.0% to 99.9% availability protection.
- 9 hours to 10 minutes of tolerable unavailability per week.

Operating model selector



A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

You Build It You Run It

For the furniture retailer, the appointments and bedroom frontends match with You Build It You Run It.

SOFTWARE SERVICE	AVAILABILITY TARGET	DEPLOYMENT TARGET	OPERATING MODEL
appointments	95.0%	Weekly	You Build It You Run It
bedroom	99.0%	Daily	You Build It You Run It

In the private sector, it's possible to have digital services with a low level of revenue exposure, and high feature demand. For example, a medical publishing company with a per-institution subscription model could have customers clamouring for new features, but a website failure would not incur direct revenue loss as customer dissatisfaction is not linked to subscription renewal. Operational costs and indirect revenue loss would need to be factored into the availability target calculation.

We'd urge caution if you believe a low availability target and a low deployment target apply to your digital services. Low financial exposure on failure and low product feature demand could signify you're building the wrong thing for your customers.

For the furniture retailer, the ecommerce-platform COTS application matches with Ops Run It at a higher availability target.

SOFTWARE SERVICE	AVAILABILITY TARGET	DEPLOYMENT TARGET	OPERATING MODEL
ecommerce-platform	99.9%	Monthly	Ops Run It

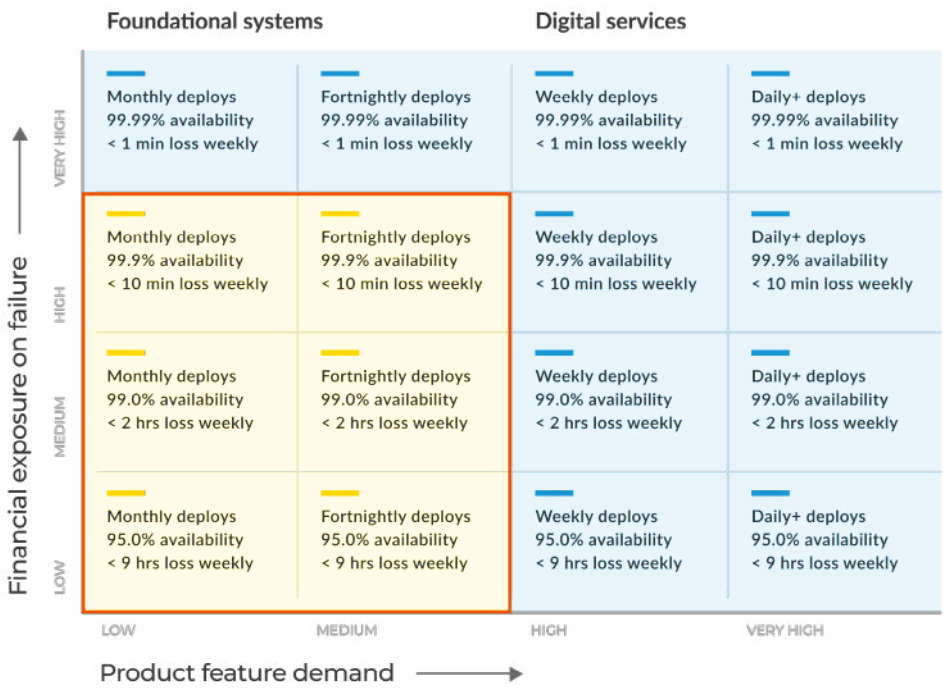
Select Ops Run It for foundational systems

Select Ops Run It for your self-hosted COTS applications and custom integrations, when your desired outcomes are:

- Monthly to fortnightly deployments.
- 95.0% to 99.0% availability protection.
- 9 hours to 10 minutes of tolerable unavailability per week.

Foundational systems usually require infrequent code changes after launch.

Operating model selector



A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

You Build It You Run It

Spotlight

Traffic lights
You Build It You Run It in retail

I worked at a large retailer in Australia, where we gradually moved teams and their services from Ops Run It to You Build It You Run It. The transition can be hard, and to ensure a pragmatic approach we introduced a traffic light system. We used it to help us decide if a service should remain in Ops Run It, or be moved to You Build It You Run It.

We tracked outcomes for each service, including its availability target and deployment frequency target, to determine where they were on the traffic lights.

- Red was for a fortnightly deployment target, or lower and meant Ops Run It.
- Green was for a weekly deployment target or higher, and meant You Build It You Run It.
- Amber was for a deployment target that lay in between. The service availability target was used as a tiebreaker. We had previously concluded 99.9% was not possible to achieve under Ops Run It, as it only permitted 44 minutes of downtime in a month. So a target of 99.9% or higher meant You Build It You Run It, and a target of 99.0% or lower meant Ops Run It.

The traffic light system meant we could look at services holistically. We could build a clear picture of how many services were in Ops Run It, how many were ready to transition into You Run It, and how many required more careful thought. It allowed us to clearly see across the digital estate where we needed to focus our efforts, on changing the operational model to gain the biggest benefit to the organisation.

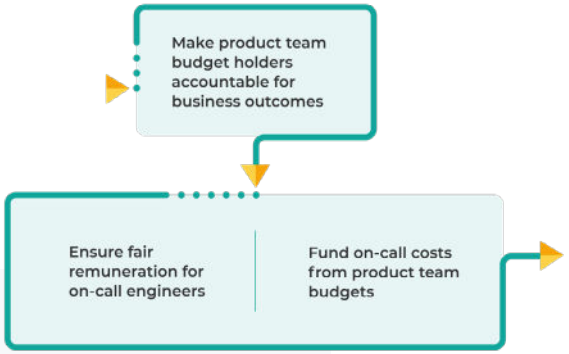
Tony Nguyen
Principal Consultant

Equal Experts
Australia

Governance

You Build It You Run It governance is a radical departure from Ops Run It. These practices are a clear, long-term commitment to You Build It You Run It. They're hard to implement, but if none of them are put in place it's unlikely your on-call product teams will be successful. Here's an approximate sequencing to implementing governance practices.

You Build It You Run It - governance practices



These practices are linked to our principles of **operating models are insurance for business outcomes** and **operating models are powered by feedback**.

Make product team budget holders accountable for business outcomes

Empower the budget holders for product teams to be accountable for deployment throughput, service reliability, and the learning culture for their digital services. The budget holder for a product team will be one of:

- Your Head of Product, if team funding comes from a product budget.
- Your Head of Delivery, if your organisation has separate Product and IT departments and team funding comes from an IT budget.

Making this change encourages the budget holders to translate business goals into operational objectives, incorporate customer feedback into development activities, and strike a balance between delivering product features and operational features. They devolve responsibilities such as availability target selection and on-call support to their product teams.

This is a radical departure from Ops Run It, in which the Head of Operations is accountable for the reliability of all software services. The Head of Operations retains accountability for foundational systems in You Build It You Run It.

We recommend this **RACI model**:

CATEGORY	HEAD OF OPERATIONS	APP SUPPORT TEAM	HEAD OF PRODUCT	HEAD OF DELIVERY	PRODUCT TEAMS
Digital services reliability • Availability • Support • Telemetry • Reporting	I	I	A for product funded services	A for IT funded services	R
Foundational systems reliability • Availability • Support • Telemetry • Reporting	A	R	I	I	I

Operability incentives are maximised for product teams when their budget holders are accountable for live support. If this practice isn't implemented, you'll suffer from the **responsible but unaccountable pitfall**.

Fund on-call costs from product team budgets

Pay for on-call support for a digital service from the product team budget. This means the product team budget holder pays for:

- *On-call standby costs.* Compensation for one or more product team members making themselves available in case of an out of hours incident. This is part of the ongoing run cost.
- *On-call callout costs.* Compensation for one or more product team members responding to an out of hours incident. This is a per-incident cost.

If the on-call opex budget is owned by your Head of Operations, transfer their budget line item for digital services into an on-call capex budget allocated to the product team budget holders. Split the on-call capex budget into a line item per digital service, and allocate each line item to the corresponding product team. This maximises incentives for each product team budget holder to:

- Establish desired business outcomes prior to any development efforts.
- Choose an appropriate availability target that balances business outcomes insurance with a run cost estimate.
- Prioritise the protection of live product functionality alongside the delivery of new product features.
- Lend their credibility to organisational changes that improve on-call experience, such as product teams automating their own telemetry toolchains.
- Incorporate graceful degradation and adaptability into the customer experience.

If on-call funding for digital services stays with your Head of Operations, you'll suffer from the **responsible but unaccountable pitfall**.

Ensure fair remuneration for on-call developers

Offer a level of pay to on-call developers for 24x7 support that recognises the inconvenience of out of hours support. We recommend choosing one of these payment models, based on your own organisational context:

NAME	REMUNERATION	COMMENTS
Standby payment only	Pay a flat rate per hour/day/week for a developer to be ready out of hours for an incident.	This model is easy to communicate and implement for cost tracking. The flat rate can be calculated based on the product lifecycle and prior incidents, or simply chosen as an arbitrary amount. See ***How we've evolved on-call at Monzo*** by Shubheksha Jalan.
Standby and callout payment	Pay a flat rate per hour/day/week for a developer to be ready out of hours for an incident, and pay a flat rate per hour for incident response.	This model has more complicated cost tracking. It needs callout payments to be carefully designed, in order to incentivise a rapid incident response.

We don't recommend callout payments only, as it's unfair remuneration. It's important to compensate your L1 on-call developers for the disruption that on-call standby causes to their lives outside of work. Always being available via phone, pager, and/or laptop out of hours has an impact on people.

Changing remuneration means updating contracts with suppliers and employees, to recognise 24x7 L1 support as a paid activity for product team developers. This can be a difficult process, and there's no one right answer on how much to pay. Transparency during this process is important, so people can understand the available funding. There's a lot of variability in how organisations approach this. See ***2019 On-call compensation survey*** by Spike Lindsey *et al*.

We've listened to senior managers in different organisations say that You Build It You Run It payment models are too expensive compared to Ops Run It, particularly if operations teams are **outsourced to a third party managed service** for lower costs. It's a flawed comparison, because operating models are insurance for business outcomes. You Build It You Run It has a more cost effective premium than Ops Run It for digital services, where higher standards of deployment throughput and service reliability are required. Ops Run It remains the right choice for COTS applications.

Your product team developers already have intrinsic motivation to be on-call. It's vital they have an extrinsic motivation as well, that recognises their contribution to your organisation out of hours. This is hard to put into place, but if it doesn't happen people will decline to do on-call and you'll be in the **limited on-call schedule pitfall**.

Spotlight

Managing on-call at John Lewis & Partners

In 2020, the product team responsible for registration and authentication at John Lewis & Partners was delivering a roadmap to decommission a legacy system, and introduce an integration with a third party identity provider.

The team began to launch features involved in the checkout process, which were dependent on the new identity provider. This meant resolving high profile incidents involving lost sales became part of their world.

Over two weeks, we met several times as a team to decide what we wanted our out of hours support to look and feel like.This included updating our runbooks, dashboards, and logging. We reviewed the incident management process, and agreed amongst ourselves how we defined an on-call shift.

The knowledge the team was the first and last line of defense for registration and authentication, at any hour of any day, was baked into every line of code. Did that mean we had no incidents? Definitely not. But when incidents were resolved, we could quickly implement any improvement actions to avoid a repeat incident.

Things that worked well included:

- Mutually supporting some team members who were more nervous about running out of hours support.
- Regularly debriefing alerts, to establish a common view of what could happen.
- Building positive relationships with the operations teams supporting the legacy systems we were replacing.
- Performing dry alert runs, to decide which alerts to configure for out of hours notifications.
- Documenting exactly how to raise an incident with a specific third party provider.

Learning how to work with the third party provider was particularly important. We figured out that asking them what made a support ticket useful would help us to get a useful response in the first instance, rather than wait for multiple delays.

Learn more about our partnership with John Lewis & Partners here.

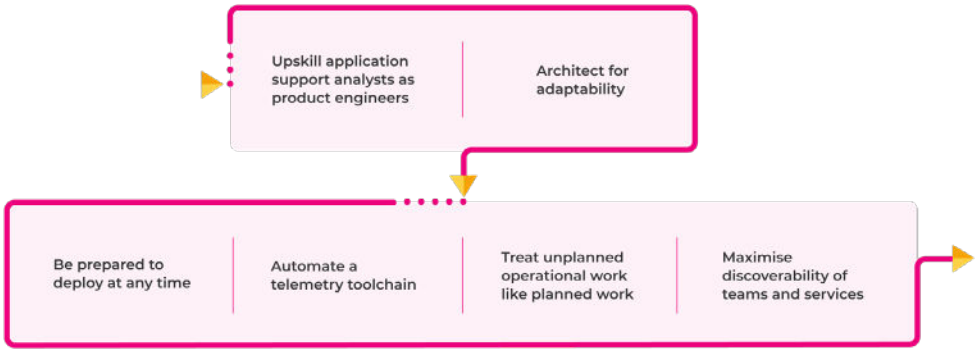
Erin O'Connor
Delivery Lead

Equal Experts
UK

Build

These practices design and build operability into digital services. Building a digital service that is easy to operate in production creates new sources of adaptive capacity, which can be utilised in abnormal operating conditions.

You Build It You Run It - build practices



These practices link back to our principle of **operating models are effective when rapid restoration comes first**. You might not need to implement all of these, but without a majority of them you'll suffer from the **excessive BAU pitfall**.

Upskill application support analysts as product team developers

If you're transitioning your digital services from Ops Run It, invest in your application support analysts and help them become product developers.

You Build It You Run It is all about incorporating operational knowledge into the design of digital services, and application support analysts are an invaluable source of such knowledge. They have a deep appreciation of operating conditions, live traffic management, and historical incidents. Upskilling analysts allows them to contribute their institutional knowledge to building operability into digital services. For example, ex-analysts with prior on-call experience can teach other product team developers how to refine alerts and minimise unnecessary callouts.

Upskilling analysts is a significant investment. We recommend creating opportunities for motivated, skilled employees interested in new career pathways. We'd adopt a gradual pace using learning materials and pair-programming, to minimise disruption for your product teams and operations teams alike. We'd encourage shadowing and knowledge sharing sessions within product teams, to avoid application support analysts being isolated as **embedded specialists**.

You Build It You Run It creates new opportunities for your application support analysts. They can remain in an operations team committed to COTS and foundational systems, and increase their technical mastery. Alternatively, they can join a product team dedicated to a digital service, and learn how to use software delivery skills to solve business problems. Some of the best product team developers we've worked with have a background in Ops Run It.

Architect for adaptability

Design loosely-coupled product teams and loosely-coupled digital services, to enable graceful degradation when a production failure inevitably occurs. This allows customers to partially use your digital services, and your business outcomes to be partially realised, while full availability is restored in the background.

Architecting for adaptability also protects an on-call product team from **excessive BAU**. Minimising service dependencies and the blast radius of anticipated failures allows a product team to prioritise availability protection over planned feature development, because there's a high standard of service reliability, which results in a low level of BAU work.

This is deeply contextual, and it varies from one digital service to another. For example, a digital service could comprise a modular monolith of many business functions, or multiple microservices per business function. Empower product teams to make their own choices, reduce failure blast radius, and remove inter-team overheads in deployment orchestration and/or incident response wherever possible.

Architecting for adaptability takes many forms:

- *Eliminate avoidable service dependencies.* Draw the bounded context of a digital service to ensure minimal dependencies on other digital services and third party systems.
- *Soften unavoidable service dependencies.* Ensure communications with downstream digital services and third party systems are via asynchronous queues, bulkheads, circuit breakers, on the wire caches, and rate limiters wherever appropriate.
- *Create availability redundancy.* Leverage cloud infrastructure to establish multi-zone availability for digital service workloads, followed by multi-region availability if necessary.
- *Set up single team ownership for digital services.* Design product and technology boundaries so one on-call product team has exclusive write access to the code repositories, configuration, and more for a digital service. One team can own multiple digital services, but a digital service can only have one owning team.
- *Establish single service ownership for databases.* Create database infrastructure so one digital service has exclusive read and write access to a relational or non-relational database. A digital service can use multiple databases, but a database instance can only have one digital service consumer.
- *Share behaviours via APIs not closed-source libraries.* Prevent the use of closed-source code libraries by mandating open-source code libraries for infrastructure code reuse, and digital services for business logic reuse. See ***Don't share libraries among microservices*** by Philip Hauer.
- *Shutter a service on failure.* Create user-friendly shutter pages to replace API and frontend entry points for a digital service for when it is entirely unavailable.
- *Shift some incident resolution onto customers.* Experiment with self-service fix actions and chatbots to introduce L0 customer support. We see this infrequently, when on-call costs have been incurred in a repeatable failure scenario that cannot easily be eliminated.

Pressure to create faster, better, cheaper digital services actively undermines efforts to increase organisational resilience. It's important to always consider long-term extensibility alongside short-term gains. See ***An interview with David Woods*** by Ipsita Agarwal.

Be prepared to deploy at any time

Ensure a digital service is always deployable. This means a monolith or a set of microservices are always passing their automated functional tests, always ready to be signed off in exploratory testing, and always prepared to go through a deployment process into production. This includes:

XP development practices. Gradually introduce pair-programming, test-driven development, Trunk Based Development, and Continuous Integration into product teams.

- ***Deployment pipelines.*** Create fully automated deployment pipelines for all activities between code commit and production deployment.
- ***Parallelised functional tests.*** Use dynamic test data and prime test runners to run suites of functional tests in parallel.
- ***Zero downtime deployments.*** Establish blue-green deployments, canary deployments, and/or dark launching to put new product features in front of customers in the day time, without any availability loss.
- ***Fast revert on failure.*** Allow a failed build, test suite, or deployment to be immediately reverted within seconds of failure detection.

The ability to deploy at any time enables a rapid incident response. As usual, we recommend a product team experiments with different approaches until the right fit is found. It's possible you won't need all of these in your organisation.

Automate a telemetry toolchain

Encourage on-call product teams to fully automate a telemetry pipeline, and continually refine their loggable, monitorable, and alertable events. Complex digital services can interact in unexpected and surprising ways. It's vital that on-call product team members can quickly identify, and diagnose abnormal operating conditions. This includes:

- *Logging.* Implement a logging stack like **Elastic** + **Fluentd** + **Kibana** (EFK) to process logs, and visualise events in dashboards.
- *Monitoring.* Install a monitoring stack like **Victoria Metrics** + **Grafana** to process metrics, and visualise business and technical events in dashboards. Commercial tools like Appdynamics and **New Relic** are also available. See ***Scaling to trillions of metric data points*** by Venkat Vaidhyanathan *et al*.
- *Alerting.* Use an alerting stack like **vmalert** + **Alertmanager** to fire alerts from logs or metrics, based on configurable alert definitions defined as code.
- *Incident response.* Use an incident response platform like **PagerDuty** or **VictorOps** to consume alerts in order to notify on-call product team members, and register incidents in a ticketing workflow system like **ServiceNow**.
- *Incident collaboration.* Use a collaboration platform like **Slack** for product team members to effectively communicate during production incidents.

We recommend a telemetry pipeline is built just like any other automated deployment pipeline. The goal is for dashboard updates, out of hours callouts, and incident ticketing to happen automatically without any human intervention.

Treat unplanned operational rework like planned work

Manage unplanned operational tasks for live digital services in a similar way to planned product and operational features. This ensures urgent operational faults can be quickly rectified, without causing too many delays to planned work.

We've heard unplanned operational rework described as 'BAU work'. Essentially, it's any operational task that hasn't been planned for by the product manager. This might include investigating an intermittent alert, fixing a deployment failure, or adding more infrastructure capacity. Manage these tasks as follows:

- *Visualise rework items.* Show unplanned operational work on the team board, alongside planned product and operational features. This could involve a separate swimlane for unplanned work items, or tagging of different work items.
- *Handle rework items.* Assign the on-call product team member in working hours to:
  - Add newly uncovered operational tasks to the product team board
  - Complete urgent operational tasks as soon as possible
  - Add non-urgent operational tasks to the backlog for prioritisation
- *Measure rework items.* Track how much time per week is spent on unplanned operational tasks, and check for an upwards trend correlating with a decline in deployment throughput and/or service reliability. For more on this, see Rework Rate measurements in ***Accelerate***, by Nicole Forsgren *et al*.

Maximise discoverability of teams and services

Visualise all your product teams and their digital services, in an information portal that any authenticated person can access. This creates a centralised source of truth for key characteristics of digital services, such as which team owns which service and measures of success.

An information portal is the backbone of alert routing, progress reporting, and urgent information lookups during production incidents. It's essential at scale, and in that scenario we build a service catalogue as part of a digital platform. See our **Digital platform playbook** by Adam Hansrod *et al*.

We're sometimes asked by our customers if there's an open-source or COTS information portal out there. We recommend a portal be built from scratch, as it's incredibly valuable to constantly curate information for an entire organisation on business metrics, reliability measures, and acquired operational knowledge.

Spotlight

Securing You Build It You Run It in retail

We worked in a large merchandise retailer, building an encryption service to protect sensitive data. This included personal data, database encryption keys, and mobile payment keys. We chose to implement You Build It You Run It because the pressure to keep our services available was very high, and our infrastructure was complex. It would have been difficult to pass our services over to a separate operations team.

We made the alert system for this infrastructure quite sensitive, so we could catch all sorts of problems. However, it didn't have the desired effect. Some alerts were too sensitive due to brief vendor instabilities, which we couldn't fix during callouts. It decreased our attentiveness to alerts to some extent, and tired out the team. After that, we concluded we should only receive alerts that are actionable.

We made the decision to move to service level objectives (SLOs) for our alerts. This benefited the team a lot, as clear availability targets and priorities reduced unnecessary stress.

Our alerts were also a great tool to improve our services. As we were investigating alerts, we were often able to quickly fix the underlying problems.

We then realised we should extend these practices to disaster recovery, which was a great tool for bringing the whole team up to the same level of understanding.

From this experience, I learnt that You Build It You Run It practices are worth investing in, as it is easier to react to a problem when you are woken up at night if you've practiced common failure patterns. When our team had new team members, we were always eager to bring them up to speed by putting them on-call.

Natalia Oskina
Developer

Equal Experts
UK

Operational Enablers

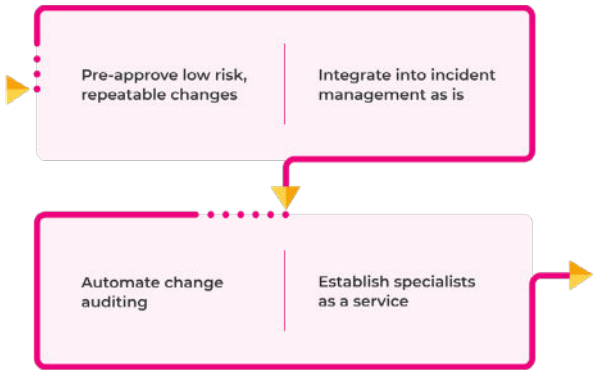
These practices allow on-call product teams to collaborate effectively with operational enabler teams. This includes:

- Integrating digital services into service management processes for change management, incident management, and more.
- Working with specialist teams such as database administrators and network administrators.

You Build It You Run It co-exists with Ops Run It, as a hybrid operating model. Product teams use the same workflows that operations teams use for COTS and foundational systems. This helps operations teams to buy into product teams managing their own digital services, and it drives consistent behaviours across your organisation.

Product teams have the incentives and technical skills to automate integration touchpoints. They collaborate with operational enabler teams to implement fully automated workflow schemes for digital services. Over time, COTS and foundational systems can be migrated to the same workflow schemes, and everyone benefits from faster, more reliable service management.

You Build It You Run It - operational enabler practices



These practices are based on our principle of **operating models are selected on financial exposure and product feature demand**.

Pre-approve low risk, repeatable changes

Establish pre-approved change requests for low risk and repeatable changes. Continue to use change approvals for high risk and/or unrepeatable changes. In ITIL v3, this is the difference between standard changes and normal changes. This ensures regular, automated production deployments performed by an on-call product team can happen in a timely fashion.

Integrate into these change management actions:

1. Allow a product manager to create a pre-approved change request template for a digital service.
2. Automatically raise a change request when the last pre-production activity is successfully completed:
    a. For a low risk and repeatable change, complete a pre-approved change request.
    b. For a high risk or unrepeatable change, send a change request to a change manager for approval.
3. Automatically check prior to a production deployment that an approved change request exists, and do not permit a deployment if none is found.
4. Automatically close the change request when the production deployment is completed.

This practice depends on a high proportion of low risk changes for a digital service, and a regular cadence of successful production deployments. Without it, you'll suffer from the **Change management treacle pitfall**.

Integrate into incident management as is

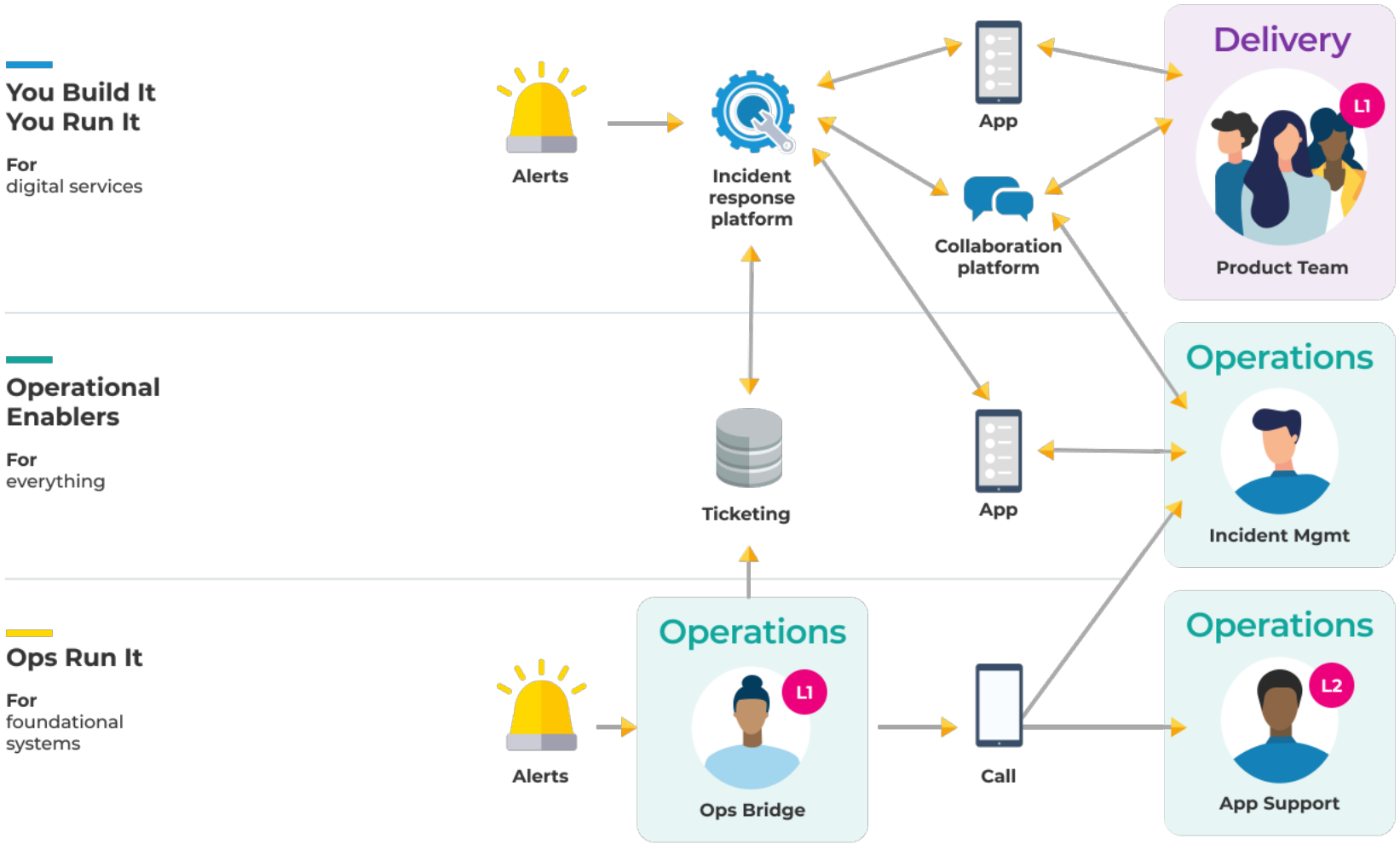
Automate incident management touchpoints into your ticketing system. This allows an on-call product team to manage their own incidents with minimal friction, and demonstrates a long-term commitment to collaborating with major incident managers.

Integrate into these incident management actions:

1. Notify an on-call product team member when an alert is fired.
2. Create an incident in the ticketing system when an alert is fired.
3. Acknowledge an incident in the ticketing system when an on-call product developer is available.
4. Reassign an incident to a different digital service, COTS application, or foundational system when necessary.
5. Notify an incident manager when an incident is declared as a major incident.
6. Update an incident in the ticketing system when in-progress notes are available.
7. Resolve an incident in the ticketing system when an alert is resolved.

We recommend paying for a SaaS incident response platform such as **PagerDuty** or **VictorOps**. An incident response platform orchestrates alert notifications for on-call product teams, which means an on-call product developer can be notified of an alert in under a minute. It can manage an out of hours schedule for incident managers as well as product teams, which means an on-call developer can quickly contact someone for advice during a major incident. It can also offer bi-directional updates between its own alerts and incidents in your ticketing system. This removes hours of manual efforts on tickets, and ensures high quality records are preserved.

Integrating into incident management as is



It takes time to integrate You Build It You Run It into the same incident management process as Ops Run It, and we've seen it pay off multiple times. We've witnessed dramatic reductions in time to acknowledge and time to resolve an incident, and improvements in ways of working between different teams. If you don't do this, you'll fall into the **No major incident management pitfall**.

Automate change auditing

Build each deployment pipeline as a fully automated compliance tool for change management. Accelerate change feedback loops from weeks or months into minutes.

Use a version control system such as **GitHub** to store all updates to:

- Alert definitions
- Code
- Configuration
- Infrastructure definitions
- Logging dashboards
- Monitoring dashboards
- Reference data

Audit in every deployment pipeline:

- The versions of your digital services currently deployed in which environments
- The updates in your version control system that can be traced back from different versions of your digital services
- The deployment timestamps, deployer names, and approved change requests for production deployments
- The deployment lead time and deployment frequency for production deployments

This is one way to implement compliance as code. Work with your change managers to understand the reports they need to satisfy internal compliance requirements. Free them up to tackle higher value work, such as orchestrating production deployments of foundational systems. Without this, you'll suffer from the **Change management treacle pitfall**.

Establish specialists as a service

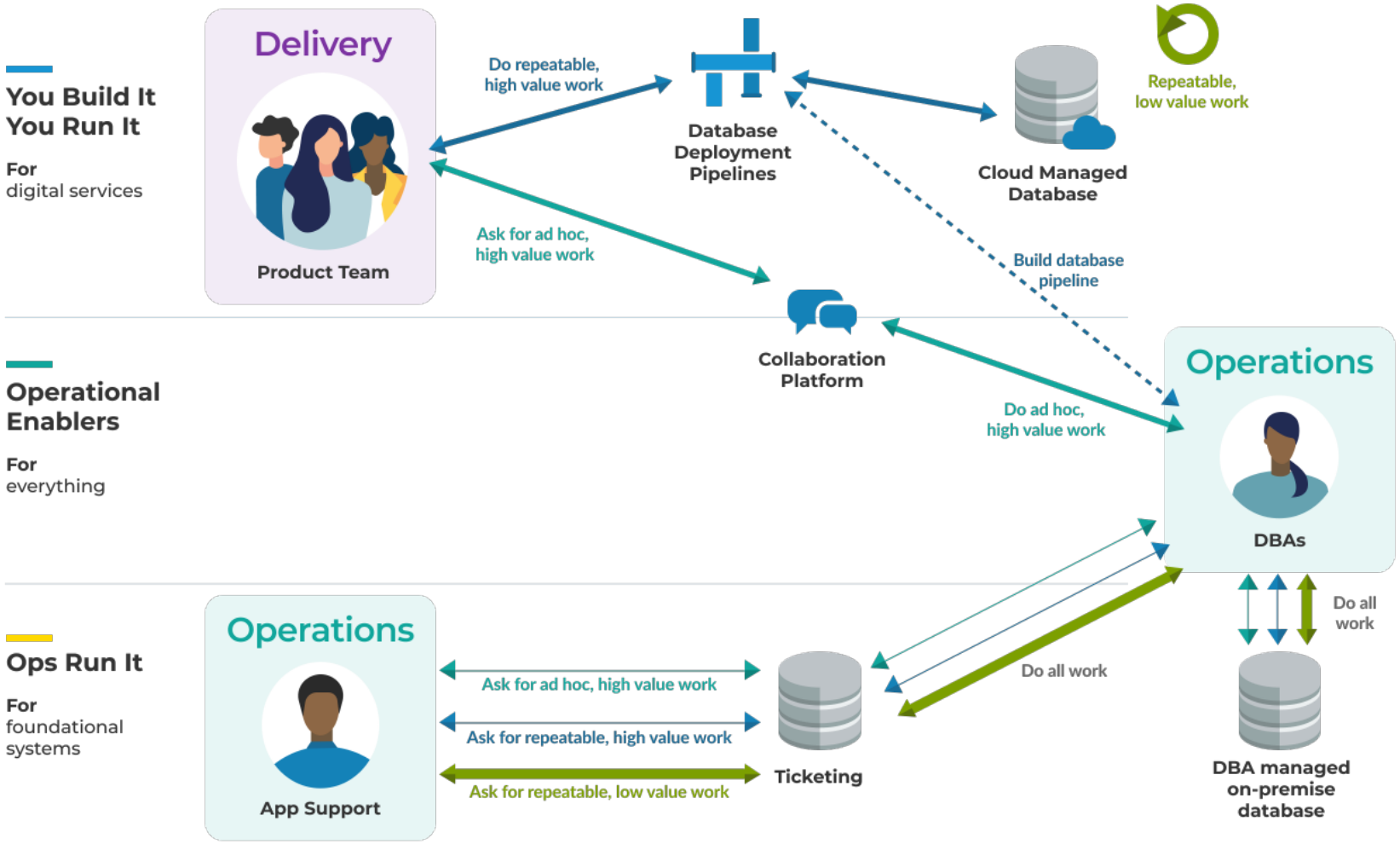
Turn the skill sets of specialised operations teams into consumable services. This creates a balance between breadth of cross-functional product teams and depth of specialist expertise in operational areas.

Some operations teams possess scarce and highly valued engineering capabilities. The usual example we see is a few database administrators managing an on-premise relational database, for multiple product teams operating digital services in a cloud provider. The answer isn't to cross-train developers, or hire more database administrators. It's to eliminate repeatable work by migrating databases into your cloud provider, and by creating self-service deployment pipelines for your product teams.

Offloading and automating database tasks frees up your database administrators to provide high value expertise on demand, such as troubleshooting live performance problems. For more on this, see **Stop trying to embed specialists in every product team** by Bethan Timmins.

If you don't implement specialists as a service, you'll endure time-consuming interactions with operational teams, and eventually fall into the **embedded specialists pitfall**.

DBA specialists as a service



Spotlight

Launching a new sales platform with ITIL standard changes

I worked in a large media company, and we were building a multi-channel sales platform. The company had a centralised change management process based on ITIL, and it wasn't suited to our aims of frequent self-managed releases.

When we started to use change requests, we found out we could use standard changes, because we had fully automated deployments and platform releases were always toggled off on deployment. We met the criteria for repeatable, low risk changes.

We then proceeded to deploy hundreds of changes to the emerging new platform under the standard change process, until we had enough functionality in production that we could turn the sales platform on. This involved some business readiness work, working closely with contact centre staff so they use the sales platform as part of the assisted sales channel. It didn't require any change approvals either.

Some months later, I bumped into a change manager, and they asked when we were going live. I advised we'd been live for months, and had hundreds of users all very happy with the new sales platform. The change manager advised standard changes didn't sound appropriate for future deployments now it was a live system, and I said we'd been continuing to deploy multiple times a week to deliver bug fixes and new features!

When I look back on that experience, I see it was my first true experience of You Build It You Run It and all the benefits. We worked closely with the product manager, with few handoffs. We had very high levels of test automation, automated deployment pipelines, and many feature flags. And all because we were responsible for our sales platform, and took that responsibility seriously.

Paul Timmins
Lead Consultant

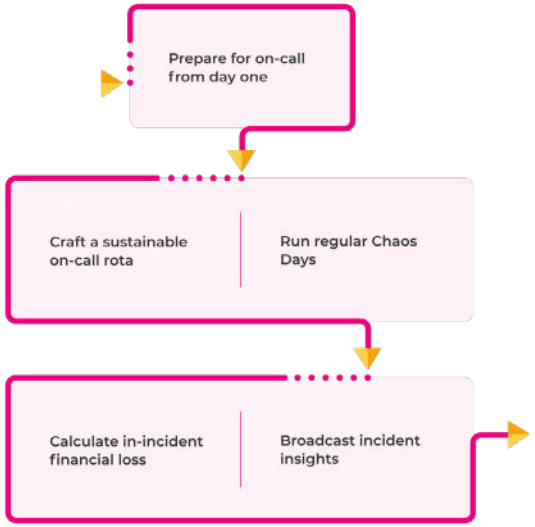
Equal Experts
Australia & New Zealand

Incident Response

These practices relate to responding to incidents, and learning from incidents. One of the drawbacks of You Build It You Run It is its **high setup cost**, and part of that is a steep learning curve for product teams. It's important they're prepared to identify, resolve, and learn from production incidents, before their first incident occurs. If these practices aren't implemented, you'll have unprepared incident responders and they'll fall into the **excessive BAU** and **limited on-call schedule** pitfalls.

These practices link back to our principle of **operating models are effective when rapid restoration comes first**.

You Build It You Run It - incident response practices



Prepare for on-call from day one

Set on-call expectations in product teams from the moment they start to build digital services. This maximises incentives for product team members to build operability into product features. It gives them time to adjust to a new mindset, in which they could be called out at 0300 to fix their own work.

On-call preparation includes:

- *Inception.* Work with the product manager and intended users to understand the required availability level, and how best to gracefully degrade upon unavailability. See our **Inception Playbook** by Neha Datt *et al*.
- *Team skills*. Balance the skillsets of different product team members - frontend developers, backend developers, testers, and more - to create enough capacity for on-call incident response later on.
- *Runbook.* Start a runbook on day one, and update it weekly with anticipated failure scenarios, failure mitigations, alert thresholds, and step by step guides.
- *Telemetry.* Update logging dashboards, monitoring dashboards, and alert definitions for every product feature. Encourage product team members to check dashboards daily.
- *Onboarding.* Set on-call expectations with prospective team members before they join. Assign telemetry and incident response permissions to new product team members as soon as they join.
- *Escalation path.* Ensure product team members have clear communication pathways for major incidents. Make it easy to nominate an incident manager as incident commander for out of hours incidents.
- *Learning mindset.* Foster a no-blame culture of collaborative learning for production incidents. Promote an emphasis on creating new knowledge for the entire organisation.

These changes can be difficult to implement when you have a tight launch deadline for a new digital service. It's important to make them a bit at a time, starting from day one.

Craft a sustainable on-call schedule

Encourage a product team to manage an out of hours on-call schedule, which safeguards the reliability of digital services and preserves work-life balance for product team members.

We recommend:

- A minimum of three product team members participating in the on-call schedule
- All product team members are encouraged to participate in the on-call schedule, regardless of role
- A default on-call rotation of one week for each on-call participant

How product team members help each other to manage the on-call schedule is vital to success. Some team members may be unable to do on-call for family or health reasons, and their decisions need to be respected. Some team members may be reluctant due to a lack of preparation, and any such problems need to be solved.

Run regular Chaos Days

Inject failures into digital services to validate they have been **architected for adaptability**. Enable product teams to hone their incident response skills, prior to any actual production incidents.

Run day-long events that introduce failures into your digital services. Encourage your product teams to reduce the blast radius of latent faults, refine their mental models, and build up relationships with dependent teams. If your production environment can't handle a controlled level of stress, run a Chaos Day in a test environment.

To find out more about running Chaos Days in large enterprise organisations, see our **Chaos Day playbook** by Lyndsay Prewer.

Calculate in-incident financial loss

Estimate the costs incurred and revenue lost during a production incident, and prioritise incident response based on that financial loss. Ensure product teams understand what is, and isn't, a major incident that requires swarming and an urgent resolution.

You Build It You Run It is based on an understanding of the financial exposure attached to a digital service. We've worked with many customers where incident financial loss is only calculated for the post-incident review, if at all. As a result, an entirely subjective incident priority is used to guide the incident response process, and there's a lot of uncertainty over what is and isn't a major incident.

Shifting that financial loss calculation left to the start of an incident clarifies how to approach incident response. It's a simple categorisation method, which avoids people wasting incident response time debating if an incident is a P1, a P2, or a P3.

We recommend calculating incident loss at the start of an incident, when costs incurred and/or revenue loss are first understood. Revise the calculation as and when new information on losses is available. That financial loss estimate can then be used to select an incident response method:

INCIDENT FINANCIAL LOSS	INCIDENT EXAMPLES	INCIDENT TYPE	INCIDENT RESPONSE METHOD
High	• Users cannot complete a user journey • Users affected by a security incident	Major	Restore availability immediately
Medium	• Users can complete a degraded user journey • Users affected by degraded performance	Minor	Restore availability by next working day
Low	• Users can complete a user journey • Users partially affected by degraded performance	Minor	Restore availability when possible, add ticket to backlog

This is always contextual, and dependent on examples. What constitutes a high financial loss per incident varies from one organisation to another.

Broadcast incident insights

Share incident insights and new discoveries with your entire organisation. Ensure on-call product teams respond to future incidents with the full knowledge of your organisation at their disposal.

You Build It You Run It encourages a growth mindset. Digital services are inherently imperfect, and a production failure implies a set of circumstances that were unanticipated or poorly understood. The only way for product teams to understand and improve what's not understood and needs improvement is to create more learning experiences.

Distribute the insights from a post-incident review to your entire organisation, as follows:

- Publish the incident report on an internal incident reporting site, and announce it on your collaboration platform e.g. Slack.
- Link out to the incident report from the incident ticket in your incident response platform, your ticketing system, and in your **information portal**.
- Host one or more internal brown bag sessions for product teams to walk through the incident as it unfolded, and discuss incident insights.
- Ensure senior leaders encourage thorough post-incident reviews, and spread knowledge themselves throughout the organisation to benefit different product teams.

Broadcasting incident insights means scarce, proprietary knowledge can reach those who need it most. The more information you share between product teams, the more collaboration and trust you'll foster.

Spotlight

The support toolchain for You Build It You Run It at a payments provider

We were engaged by one of the leading EMEA payments providers to build a payment gateway which could be white labelled for each tenant. The payments gateway was launched in multiple countries, with different payment integrations for different tenants.

The product team responsible for building the payments gateway was also responsible for running on-call support. Understanding the product features helped us to surface and monitor the right metrics. We also built an extensible alerting system, which could be configured for each tenant. When a new tenant was onboarded, a default alerting profile was assigned to them. The alerting profile was then gradually updated, based on the transaction volume and payment features opted into by the tenant. This helped to avoid false alerts.

Since the same team was responsible for building and running the payments gateway, we iterated on metrics and alerts as part of every feature cycle. We ensured every request was traced across services and external integrations. Dashboards were built for every alert, to reduce the response time for them. When the alerts were triggered, the product team was notified along with the right dashboards. Being on-call gave us an accurate context to prioritise and fix errors as they occurred.

Anant Pal
Lead Developer

Equal Experts
India

Measurement

These practices measure an operating model to decide if it's worthy of further investment. The cost effectiveness of business outcome protection is an informative, actionable measure of an operating model.

Leading and trailing measures can be used to gauge progress in deployment throughput, service reliability, and learning culture. It's important to select measures that are holistic, actionable, and do not create the wrong incentives. See ***Measuring Continuous Delivery*** by Steve Smith.

You Build It You Run It - measurement practices

These practices are linked to our principles of **operating models are insurance for business outcomes** and **operating models are powered by feedback**.

Measure deployment throughput

Measure the deployment throughput of digital services as:

MEASURE	TYPE	DESCRIPTION	SUGGESTED IMPLEMENTATION
Loose Coupling	Leading	Is a product team able to change a digital service without orchestrating simultaneous changes with other digital services	Inspect deployment history. Calculate how many deployments of different digital services occurred in lockstep
Pre-Approved Changes	Leading	Does a product team have permission to pre-approve its own change requests for low risk, repeatable changes	Inspect ticketing system. Calculate how many digital services have pre-approved change request templates
Deployment Lead Time	Trailing	How long does a release candidate for a digital service take to reach deployment	Instrument deployment pipeline. Calculate days from production deployment back to build date
Deployment Frequency	Trailing	How often are digital services deployed to customers	Instrument deployment pipeline. Calculate rate of production deployments

Measure service reliability

Measure the reliability of digital services as:

MEASURE	TYPE	DESCRIPTION	SUGGESTED IMPLEMENTATION
Failure Design	Leading	Does a digital service include bulkheads, caches, circuit breakers	Survey product teams. Are they confident they can cope with downstream failures
Service Telemetry	Leading	Does a digital service have custom logging, monitoring, and alerts	Survey product teams. Are they confident they can quickly diagnose abnormal operating conditions
Availability	Trailing	What is the availability rate of a digital service, as a Nines Of Availability e.g. 99.9%	Instrument digital services. Calculate percentage of requests that are fully or partially successful
Time To Restore Availability	Trailing	How long does it take to re-attain an availability target once it has been lost	Instrument digital services. Calculate minutes from loss of availability target to re-attainment of availability target
Financial Loss Protection Effectiveness	Trailing	What amount of expected financial loss per incident is protected by a faster than expected Time To Restore Availability	Instrument digital services and incident financial losses. Calculate maximum incident financial loss, up to time to restore allotted by availability target. Calculate percentage of maximum incident financial loss protected by a time to restore faster than allotted by availability target

Financial loss protection effectiveness is a measure we've used with customers to gauge the cost effectiveness of You Build It You Run It at scale. It's a check of projected financial loss per incident that is unrealised, because the actual time to restore is faster than the projected time to restore. It's a comparison that can be made between digital services, product teams, and even operating models.

In the **earlier furniture retailer example**, there was a furniture retailer example with a bedroom digital service. It had a maximum financial exposure of $200K per hour, and a **99.0% availability target**.

SOFTWARE SERVICE	MAXIMUM FINANCIAL EXPOSURE IN AN HOUR	AVAILABILITY TARGET	TOLERABLE UNAVAILABILITY IN A WEEK
bedroom	$200K	99.0%	1:40:48

Assume the bedroom service is unavailable for 30 mins, and the incident financial loss is calculated as $100K. The tolerable unavailability per week for 99.0% is 1 hour 41 mins, which would have produced a $336K incident financial loss. Financial loss protection can therefore be calculated as 70% of a projected incident financial loss was unrealised, as $236K of a theoretical $336K was protected by a faster than expected time to restore.

SOFTWARE SERVICE	INCIDENT DURATION	INCIDENT FINANCIAL LOSS	INCIDENT FINANCIAL LOSS AT TOLERABLE UNAVAILABILITY PER WEEK	INCIDENT FINANCIAL LOSS PROTECTION	INCIDENT FINANCIAL LOSS PROTECTION EFFECTIVENESS %
bedroom	0:30:00	$100K	£336K	£236K	70%

For more on measuring financial loss protection effectiveness, see our case study on **How to do digital transformation at John Lewis & Partners**.

Measure learning culture

Measure the progress of a learning culture as:

MEASURE	TYPE	DESCRIPTION	SUGGESTED IMPLEMENTATION
Post-Incident Review Readers	Leading	How many readers does a post-incident review for a digital service have	Instrument post-incident reviews. Calculate how many unique visitors each review receives
Chaos Days Frequency	Leading	How often are Chaos Days run for a digital service	Instrument Chaos Day reviews. Calculate rate of review publication
Improvement Action Lead Time	Trailing	How long does an improvement action from a post-incident review for a digital service take to be implemented	Instrument post-incident reviews. Calculate days from improvement action definition to completion
Improvement Action Frequency	Trailing	How often are improvement actions implemented for a digital service	Instrument post-incident reviews. Calculate days between implementation of improvement actions

Trends in post-incident reviews are of particular interest. Usage in training materials and citations in internal company documents could also be included. See **Markers of progress in incident analysis** by John Allspaw.

XINJA

Spotlight

You Build It You Run It on the Xinja banking platform

This customer was a fintech startup, with very limited money and skilled employees. It was a greenfield project in every sense, including business processes as well as technical implementation. I worked on the single technical team responsible for delivery, and therefore the onus of the entire system was on us. It was the classic case for You Build It You Run It.

To achieve this, automation was key. Adoption of Continuous Delivery enabled us to focus our energy on development, not deployments. All our changes were tested and deployed through a single automated pipeline, for simplicity and ease of tracking. Operational tasks needed to be in code as well, to allow for seamless deployments of infrastructure and application changes.

Monitoring and system reliability were at the heart of the system design as well, since reduced issues meant more time for development. Code quality and system health awareness checks also steered the interests of our team.

Since the business and operational knowledge was contained within our team, the customer only needed to contact us to raise their issues. This meant issues were resolved rapidly, and learnings from the issues were fed back into the design, testing and monitoring of the system.

Learn more about our partnership with Xinja Bank here.

Adrian Ng
Lead Developer

Equal Experts
Australia

Scale

These practices explain how to scale You Build It You Run It across a large organisation with many product teams and many digital services. Without these practices, you'll suffer from the **Linear run cost pitfall**.

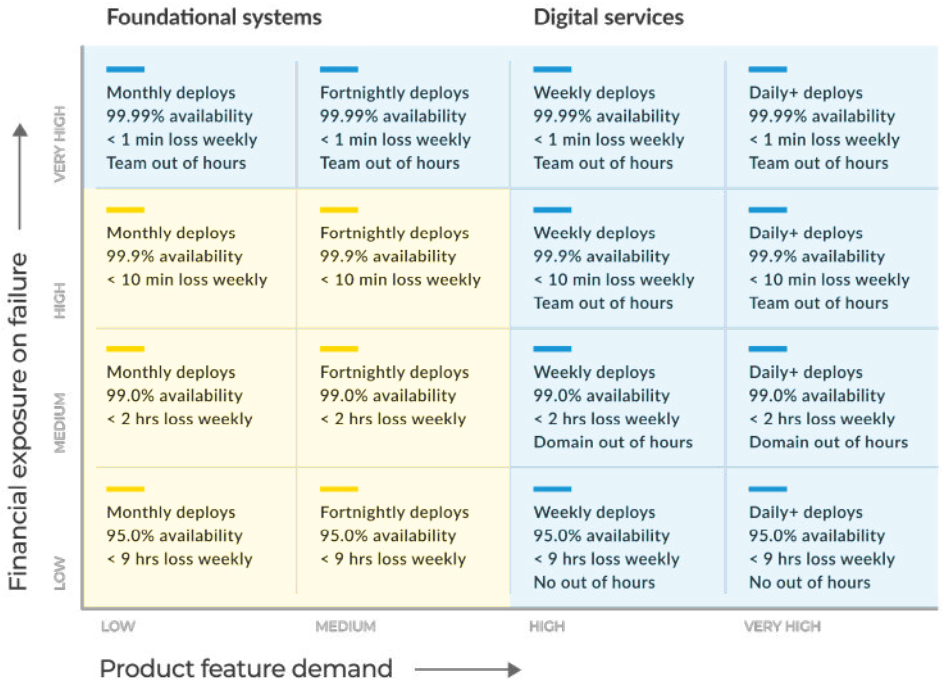
At Equal Experts, we don't believe in prescriptive scaling frameworks. We believe in applying the same holistic principles and practices for deployment throughput, service reliability, and learning culture to 1, 10, or 50 product teams, in a cost-effective way.

This means:

1. Use the same You Build It You Run It **principles** and **practices** as before.
2. Implement a **digital platform** to flatten setup costs for product teams and digital services. See the ***Digital Platform Playbook*** by Adam Hansrod *et al*.
3. Introduce additional selection practices for finer-grained operating model choices.

As the number of your product teams and digital services scales up, there can be a temptation to centralise some incident response in a new operations team. It's important to resist this idea, because it's just another form of Ops Run It that will damage delivery throughput, service reliability, and learning culture.

Operating model selector at scale

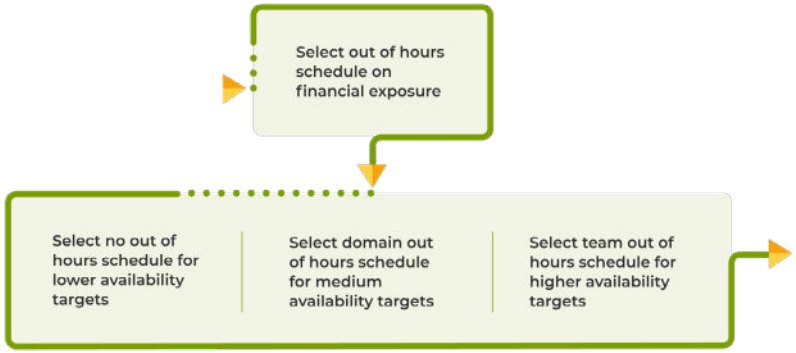


A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

You Build It You Run It

You Build It You Run It - scale practices



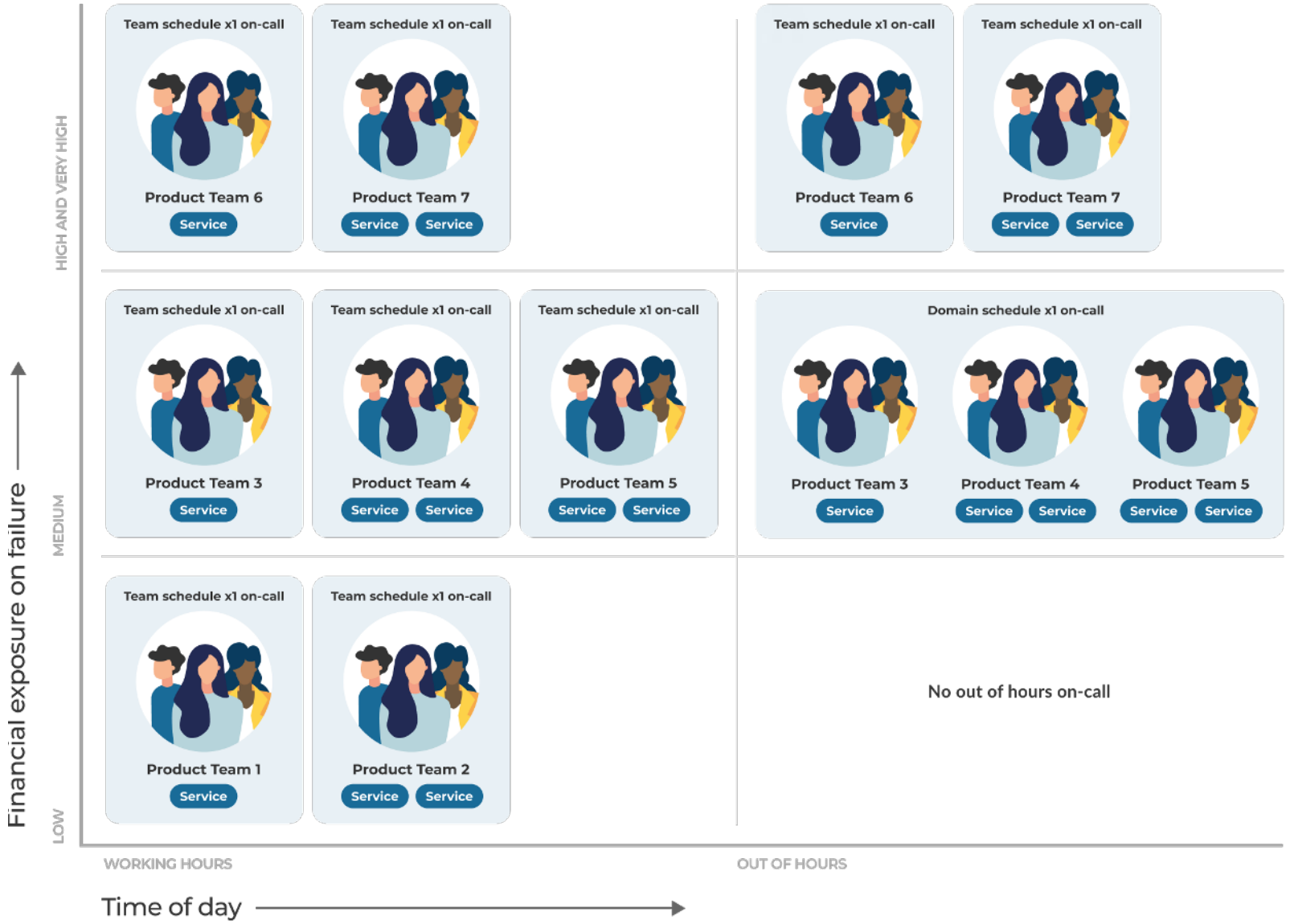
These practices are linked to our principles of **operating models are insurance for business outcomes**, and **operating models are selected on financial exposure and product feature demand**.

Select out of hours schedule on financial exposure

Ensure that **availability targets are selected on financial exposure**, and match an on-call level to an availability target. This achieves a balance between financial exposure, run costs, and remuneration for on-call product teams developers that doesn't weaken operability incentives.

You Build It You Run It protects business outcomes. It doesn't mean every digital service needs 24x7 on-call support. Not every digital service needs to be 99.99% available, and always on. Some digital services have a low level of financial exposure and don't need out of hours support, some have a medium level of exposure warranting some out of hours support, and some have a high level of exposure that justifies dedicated out of hours support.

You Build It You Run It at scale



When an availability target is selected for a digital service based on its financial exposure, it is assigned a level of on-call support in addition to its tolerable downtime per week.

In our **Selection practices**, there's a furniture retailer example with a third party COTS ecommerce platform, custom bedroom frontend, and a custom appointments frontend. The financial exposure bands linked to different availability targets can be updated to include levels of on-call:

MAXIMUM FINANCIAL EXPOSURE IN AN HOUR	AVAILABILITY LEVEL	TOLERABLE UNAVAILABILITY IN A WEEK	ON-CALL SCHEDULE IN WORKING HOURS	ON-CALL SCHEDULE OUT OF HOURS
$7,000	95.0%	8:24:00	Team	None
$250,000	99.0%	1:40:48	Team	Domain
$800,000	99.9%	0:10:05	Team	Team

In working hours, a product team always has a team schedule, and is accountable for the reliability of its digital services. Out of hours there could be no schedule, a domain schedule shared between teams, or a team schedule again.

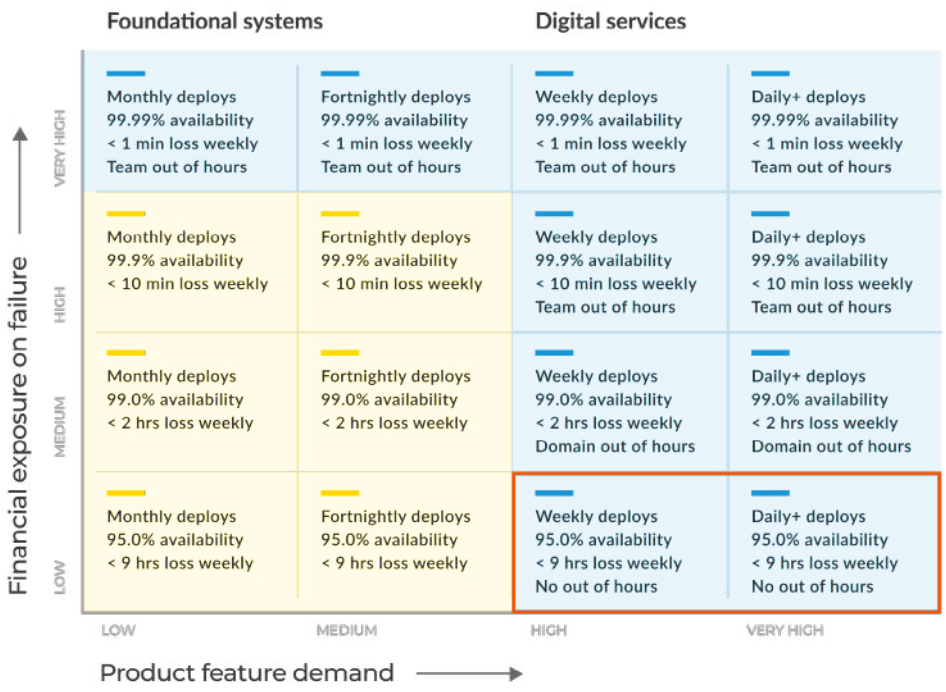
Select no out of hours schedule for lower availability targets

Don't do out of hours on-call for your digital services, when you have these desired outcomes:

- Weekly to daily deployments, or more.
- 95.0% availability protection.
- 9 hours of tolerable unavailability per week.

Reduce on-call standby costs while incentivising product teams to care about operability by promoting working hours ownership and on-call.

Operating model selector at scale



A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

You Build It You Run It

If a production incident happens during working hours, there is an immediate callout to the owning product team, and they respond to the incident based on their **in-incident calculation of financial loss**. If an incident happens out of hours, the callout is suppressed until the start of the next working day. This incentivises a product team to build operability into a digital service without out of hours support, in order to avoid a production incident spilling over into the next working day.

SOFTWARE SERVICE	MAXIMUM FINANCIAL EXPOSURE IN AN HOUR	AVAILABILITY TARGET	ON-CALL SCHEDULE IN WORKING HOURS	ON-CALL SCHEDULE OUT OF HOURS
appointments	$6,000	95.0%	Team	None

It's important to protect operability incentives for product teams who are only on-call during working hours. If your organisation has Ops Run It for foundational systems, ensure that digital services cannot be covered out of hours by that operating model.

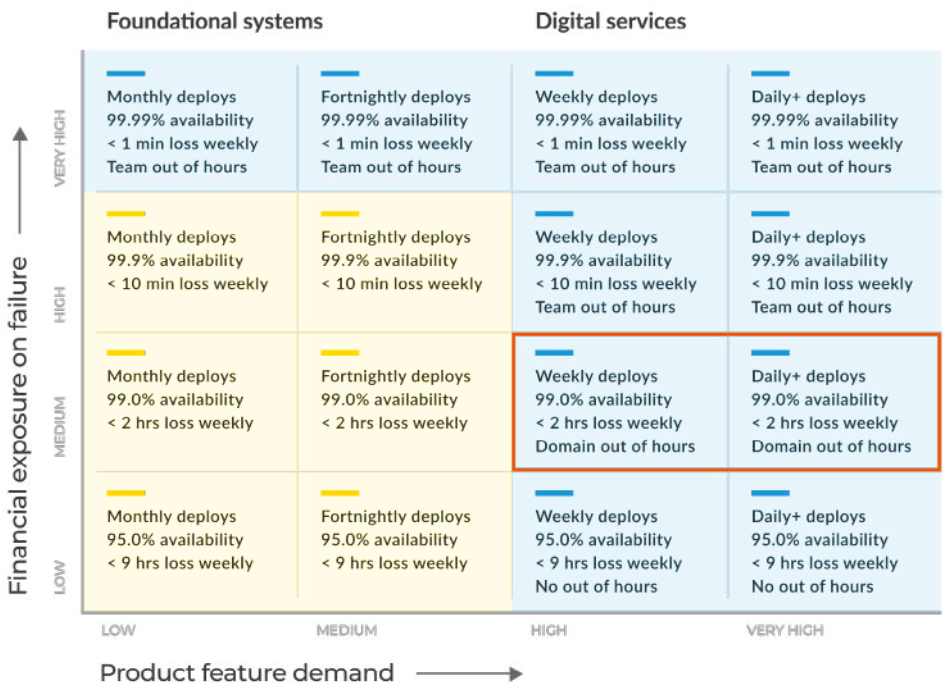
Select domain schedule out of hours for medium availability targets

Share out of hours on-call schedules between sibling product teams for your digital services, when you have these desired outcomes:

- Weekly, daily, or more frequent deployments.
- 99.0% availability protection.
- 2 hours of tolerable unavailability per week.

Reduce on-call standby costs while incentivising product teams to care about operability, even when they're on-call infrequently.

Operating model selector at scale



A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

You Build It You Run It

A domain schedule is a logical grouping of digital services, with an established affinity. The owning product teams are considered to be siblings. The domain construct needs to minimise on-call cognitive load, simplify knowledge sharing between teams, and focus on business outcomes. We recommend either of the following:

- Product domains grouped by customer journey
- Architectural domains grouped by technology capabilities

We don't recommend geographic domains grouped by region, or technology domains grouped by tool choices. One out of hours schedule for all digital services in a country, or all digital services that use Kotlin produces a mishmash with cross-cutting product boundaries and a high cognitive load. This has a negative impact on the time to diagnose and resolve production incidents.

For the furniture retailer, the bedroom frontend has a 99.0% availability target, which matches to domain out of hours on-call. The owning product team has to identify other digital services in the same product domain, and work with those product teams to establish a shared domain on-call schedule.

SOFTWARE SERVICE	MAXIMUM FINANCIAL EXPOSURE IN AN HOUR	AVAILABILITY TARGET	ON-CALL SCHEDULE IN WORKING HOURS	ON-CALL SCHEDULE OUT OF HOURS
bedroom	$200,000	99.0%	Team	Domain

Domain schedules balance strong operability incentives with run costs. They aren't perfect, and have their own complications:

- ***Domain on-call funding.*** Budget holders for different product teams in the same domain need to choose one of them as the sole budget holder for on-call funding.
- ***Domain knowledge synchronisation costs.*** The cost of sharing knowledge about multiple digital services between multiple product teams can be high. Knowledge sharing across a business context can be difficult. Knowledge sharing across technology choices can be reduced, if teams are encouraged to use the same tools for their digital services wherever possible.
- ***Domain affinity.*** Product teams may disagree on which digital services comprise a domain, or wish to run their own team schedules regardless of availability targets.

These can be mitigated by ensuring product teams are aware of domains and on-call responsibilities from day one. The sooner a product team is aware they're due to share a domain schedule with another team, the easier that eventual process will be.

Select team out of hours schedule for higher availability targets

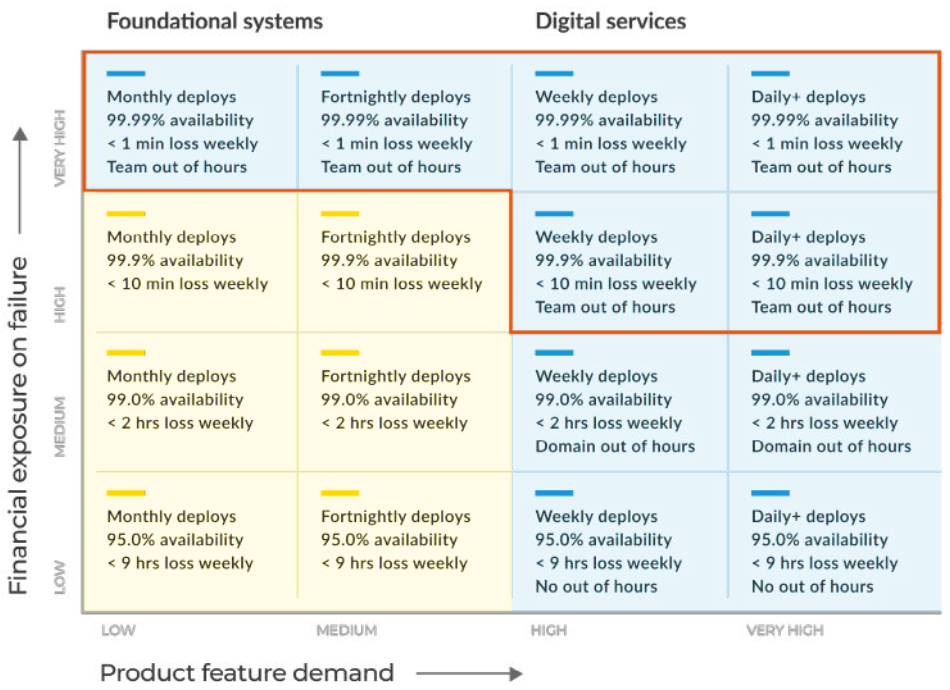
Do out of hours on-call at scale for your digital services, when you have these desired outcomes:

- Weekly to daily deployments, or more.
- 99.9% or more availability protection.
- 10 mins or less of tolerable unavailability per week.

Maximise incentives for product teams to care about operability.

This is no different from the standard You Build It You Run It. There should be a small number of digital services that require this on-call level. A high number of product teams with their own team schedules is a weak signal that something is wrong in availability target selection.

Operating model selector at scale



A product manager selects an option based on financial exposure, product feature demand, and effort. Each option has a deployment target, an availability target and a tolerable downtime a week. Moving up or right to select an option increases the implementation effort.

Ops Run It

You Build It You Run It

Spotlight

Domain schedules at John Lewis & Partners

I work on the digital platform team at John Lewis & Partners. We've built a digital platform that allows product teams to do You Build It You Run It as easily as possible. It's part of the reason why John Lewis & Partners won a digital platform award!

In late 2019, we already had 30 digital services, and another 30 were forecast for a year later. We knew having one person on-call per digital service wouldn't be cost effective, especially for digital services that aren't business critical out of hours.

Before development of a new digital service begins, the platform team has a non-technical onboarding conversation with the product manager and architect for the digital service. We have to agree on:

- Availability target. We use revenue and brand as the primary indicators of business value, and financial exposure.
- On-call level. We select an on-call level based on the availability target. We've implemented no on-call for digital services at 95.0% availability, domain on-call schedules for 99.0% availability, and team on-call schedules at 99.9%.
- Product domain. The business domain that owns the digital service. This forms the basis for a domain on-call schedule at 99.0% availability.

Availability target and on-call level aren't set forever. Once a digital service is live, they can vary based on trade patterns, day of the year, and incremental revenue.
A majority of our digital services fall into the bracket for domain schedules.

A digital service is covered overnight by a developer from the owning product team, or a sibling team in the same product domain. This incentivises product teams to focus on operability even though they don't have the highest availability target. It spreads product knowledge between teams, and establishes a common language around the domain and upstream/downstream dependencies. Most importantly, it's cost effective, as the number of people on-call does not scale linearly with teams.

We use PagerDuty as our incident management platform. We've automated the creation and grouping of digital services in PagerDuty, based on their on-call level and product domain. If you work on a product team that's part of a domain schedule, all you have to do in PagerDuty is find the entry for the domain, and add in your overnight support. All the administration is done for you.

Learn more about our partnership with John Lewis & Partners here.

Yasin Kothia
Lead Consultant

Equal Experts
UK

Pitfalls

These are the pitfalls we've encountered when implementing You Build It You Run It. Each pitfall drains confidence in the operating model, and ultimately puts it in jeopardy.

It's particularly important to guard against these pitfalls if your organisation previously migrated away from Ops Run It. A predisposition to central operations teams may still exist in your organisation. If one or more pitfalls have a negative impact, your senior leadership may see a return to Ops Run It as a quick, cost efficient way to safeguard the reliability of digital services.

These pitfalls aren't in any particular order.

Linear run cost

This looks like:

- ***Run cost directly linked to product team count.*** Your product teams each have a product team member on-call out of hours, and each one impacts on-call funding.

This pitfall happens when you have an on-call standby cost model of $*R*(*n*), where *R* is the remuneration rate per person and n is the number of teams on-call. As you increase your number of product teams, your on-call standby costs increase in kind. 1 team costs $*R*(*1*), 10 teams costs $*R*(*10*), and 20 teams costs $*R*(*20*).

A linear run cost can't happen with Ops Run It. One app support analyst on-call for all your foundational systems produces a cost model is $*R*(*1*). In addition, the remuneration rate per person *R* can be lower, if your app support team is outsourced.

On-call standby costs are a key operating model cost, and a linear run cost creates a perception that You Build It You Run It is too expensive. If you also suffer from the **Responsible but unaccountable pitfall**, on-call funding is a line item in a strained opex budget and it shows linear growth.

You Build It You Run It has a **risk of high run costs**, but it's not inevitable. Furthermore, run cost alone is a flawed comparison of You Build It You Run It and Ops Run It. Operating models are **different insurance policies for different business outcomes**, and their costs are multi-faceted. The **high opportunity costs** of Ops Run It are unacceptable for digital services.

To avoid this pitfall:

- **Select availability targets on financial exposure**, so each product team balances availability with engineering effort
- **Select out of hours schedules on financial exposure**, so product teams at scale do not produce linear run costs

Responsible but unaccountable

This looks like:

- *Inflated availability targets.* Your product managers are tempted to select the maximum availability target and highest on-call level.
- *Low priority operational features.* Your product managers have little reason to prioritise operational features alongside product features.
- *Weak operability incentives.* Your product teams have low motivation to constantly build operability into digital services.
- *On-call funding pressure.* Your operations manager comes under pressure to cut corners on on-call spend, whenever their opex funding is scrutinised.

This pitfall happens because your senior operations manager is still accountable for the reliability of digital services. Your product teams may feel some responsibility for their digital services, but they are not held accountable by senior leadership for production incidents. In addition, their on-call funding comes from an opex budget owned by the operations manager.

Keeping accountability away from product teams dilutes operability incentives all round. Product managers select whatever availability target they want, because someone else is paying their on-call costs. Product teams cut corners on designing for adaptive capacity, because they won't be questioned about inoperability later on.

To avoid this pitfall:

- **Make product team budget holders accountable for business outcomes**, so product teams have sole accountability for their digital services.
- **Fund on-call costs from product team budgets**, so product teams have to fund on-call themselves.

Excessive BAU

This looks like:

- *Slow delivery of planned work.* Your product teams don't deliver in a timely fashion the product and operational features prioritised by product managers.
- *High number of callouts.* Your product teams spend most of their time fighting operational problems e.g. intermittent alerts, deployment failures, infrastructure errors.
- *Low morale.* Your product teams are unhappy with their constant stream of operational problems, and inability to deliver planned features on time.

Unplanned operational work doesn't count towards planned product or operational features. We've heard this expressed as "BAU work". Day to day operational tasks simply need to be completed by a product team, and they constitute rework. If the amount of BAU work is excessive, a product team cannot complete planned work when expected by the product manager.

This pitfall happens when digital services cannot gracefully degrade on failure, when production deployments aren't consistently applied in all environments, and when the telemetry toolchain isn't fully automated. Excessive BAU can be hard to detect, because product teams don't always track operational rework in a ticketing system like Jira. Product team members might fix intermittent alerts, deployment failures, infrastructure errors etc. without a ticket.

It can be spotted by measuring the percentage of time product team members spend on planned work, each week.

To avoid this pitfall:

- **Treat unplanned operational rework like planned work**, so operational rework can be tracked and managed like any planned work
- **Re-architect digital services for adaptability**, so graceful degradation on failure is possible
- **Create a fully automated deployment pipeline**, so failed deployments are minimised and can be quickly reverted
- **Establish an automated telemetry toolchain**, so dashboards and alerts are reliable and can be updated at any time

We don't see this pitfall in practice as often as it's feared. We believe the fear of it comes from the historic levels of operational rework incurred for on-premise software services. For cloud-based digital services, a lot of operational changes are automatically handled by the cloud provider.

Change management treacle

This looks like:

- *Insufficient deployment frequency.* Your product teams can't achieve weekly deployments or more.
- *Prolonged change management process.* Your change management team insists each deployment completes a time-consuming change management process.

We refer to slow, prolonged change management processes as treacle, and they have a significant impact on deployment lead times. It naturally encourages fewer, larger deployments, which makes it harder to understand the changeset and diagnose any production problems.

This pitfall happens when your change management process is entirely reactive, a single change category is applied to all production deployments, and each deployment requires a change approval. It creates a fraught relationship between your change management team and your product teams.

At the same time, it's important that product teams comply with internal requirements on change management, particularly if your organisation follows IT standards such as ITIL v3. A discussion with a change management team to streamline change approvals needs to be accompanied by a commitment to preserve change auditing.

To avoid this pitfall:

- **Pre-approve low risk, repeatable changes** to accelerate a majority of deployments
- **Automate change auditing** for compliance with change management processes

No major incident management

This looks like:

- *Lack of incident response collaboration.* Your product teams don't know how to involve other product teams and/or operational enabler teams in major incidents
- *Major incident ignorance.* Your incident management team don't know when digital services are experiencing major incidents
- *No crisis communications.* Your senior leadership don't know when major incidents are creating significant financial losses

A lack of incident management creates inconsistent behaviours and communication pathways during major incidents. It has a negative impact on resolution times and financial losses incurred, when an incident requires more than one product team to be resolved.

This pitfall happens when your incident management team is excluded from the incident response process for digital services. It means different product teams will have distinct behaviours and communication methods during production incidents. It creates an impression that product teams aren't rigorous during incident response.

Crisis communications are particularly important during a major incident, and product teams won't know who to contact or how often to contact them with incident updates. A lack of clear, timely information to senior leadership during a major incident is an easy way to create doubts about an entire operating model.

To avoid this pitfall:

- **Integrate into incident management as is**, to ensure incident managers can be incident commanders for digital services as well as foundational systems.

Embedded specialists

This looks like:

- ***Embedded specialists shared between product teams.*** Your organisational model calls for *N* embedded specialists, each dedicated to one of your *N* product teams, but you've got less than *N* specialists and they're assigned to multiple teams.
- ***Unpredictable workload for embedded specialists.*** Your embedded specialists are either bored from a lack of work, or burned out from too much work across multiple teams.
- ***Loneliness for embedded specialists.*** Your embedded specialists don't have opportunities to work together, learn from one another, or even talk to one another.

This applies to any technology specialty tied to operational work - DBAs, InfoSec analysts, network admins, operability engineers, and more. Balancing breadth of cross-functional product teams and depth of specialist expertise is hard.

This pitfall happens as a countermeasure, after a small, central team of specialists can't handle demand from your growing number of product teams. There's little desire for developers to debug Postgres in production or learn Terraform on the job, so the answer is to embed a specialist in each product team.

However, there's a scarcity of affordable specialists in the marketplace, and the need for expertise in product teams fluctuates. The result is a large expertise bottleneck is split into multiple expertise bottlenecks. Instead of spending hours or days waiting on a central specialist team, a product team waits for hours or days for its own embedded specialist to be available.

To avoid this pitfall:

- **Establish specialists as a service**, so repeatable, common tasks are automated as self-service functions, and specialists in a central team are freed up to offer ad hoc expertise on demand.

Limited on-call schedule

This looks like:

- Product teams have a minority of team members in out of hours on-call schedules
- Product team members participating in on-call have significant disruption to their personal lives, and are on the verge of burnout

Each product team with a limited on-call schedule has digital services at risk of lengthy incident resolution times out of hours. If product team members need time off work to cope with burnout, team morale will suffer and planned product features will take longer to complete than expected.

This pitfall happens when product team members feel unprepared for on-call support, are unhappy with their remuneration, or burn out from too much time on-call over a period of time. It's important to respect the circumstances and decisions of different product team members.

To avoid this pitfall:

- **Prepare for on-call from day one**, so product team members are well equipped to handle out of hours production alerts.
- **Re-architect digital services for adaptability**, so digital services don't require substantial human intervention, and are fast to fix on failure.
- **Ensure fair remuneration for on-call developers**, so product team members feel compensated for the disruption to their personal lives.
- **Craft a sustainable on-call schedule**, so no one product team member spends too much time on-call out of hours.

This pitfall affects operations teams as well. Your app support team may have some useful organisational context and experiences to contribute.

Spotlight

Lost accountability in retail

I worked on a team at a retail customer, building a shifts app for staff mobile devices in bricks and mortar stores. We built a cloud-based platform in Azure and the mobile app with You Build It You Run It, and the benefits were clear to our customer sponsor. We achieved a time to repair of less than 10 minutes, and we deployed on average twice a day for over six months.

Unfortunately, once the shifts app became a live service and its user base hit a critical mass, its ownership changed to a different business function within the customer organisation. Some people in this new business function were unwilling to embrace our ways of working.

An **operations manager became accountable** for the reliability of our shifts app. We remained responsible for supporting the app, but we were no longer permitted to perform deployments without Change Advisory Board approval. As a result, our time to repair non-critical issues increased from 10 minutes to 1 week, of which 5 days was the governance process.

By the time I left the engagement, we were down to deploying once a fortnight. It was strange, because people celebrated how quickly we could resolve live issues, and yet they didn't quite make the connection to how important You Build It You Run It had been.

Julia Wilson
Lead Consultant

Equal Experts
UK

Resources

Further resources on You Build It You Run It are available, from our customers and from within Equal Experts.

This includes case studies, conference talks, and in-depth articles. The featured customers include John Lewis & Partners, HMRC, and more.

Please visit http://www.equalexperts.com/our-services/deliver/you-build-it-you-run-it to learn more.

Contributors

We'd like to thank everyone within Equal Experts who has shared their wisdom and experiences with us, and have made this playbook possible.

Adam Hansrod
Abby Bangser
Adrian Ng
Ali Lotia
Alun Coppack
Anant Pal
Andy Canning
Ann Yong
Beccy Stafford
Brian Mason
Cassiano Leal
Charles Tumwebaze
Dan Mitchell
Dave Hewett
David Cox
Donald Graham
Erin O'Connor
Jon Barber
Jon Dickinson
Julia Wilson
Harish Kannarao
Kev Roberts
Liz Leakey
Lyndsay Prewer
Natalia Oskina
Ogonna Iwunze
Paul O'Donnell
Paul Timmins
Paulo Lai
Raj Wilkhu
Reda Hmeid
Thomas de Cad'oro Granier
Tony Nguyen
Yasin Kothia

About Equal Experts

We are a consultancy of senior practitioners. **Experts in the design, delivery, and scaling of custom software products and data.** We work with customers to bring the step change they need to sustain a culture of digital innovation.

We help customers to make better decisions with data. Use design thinking to unlock value faster. And we accelerate the pace of delivery as they scale.

We are a global network of over 2,000 people, based in 5 continents. From day one, we have selected only senior technologists. People who want to stay hands-on and deliver valuable software. Experts who want to work with other experts, so they can keep learning from each other.

Our network is made up of permanent staff and independent contractors. Each selected following stringent processes. Each valued equally.

Being low ego, grown-up experts with intrinsic motivation to learn and grow means we can minimise hierarchy. This frees consultants to focus on the needs of their customers. We harmoniously blend culture and capabilities from both organisations. This creates a flywheel effect that makes innovation easier over time.

What sets us apart is our Expertise. It works because we treat each other and our customers as Equals. This is what makes us Equal Experts. This is how we bring the step change needed to sustain innovation.

If you'd like to learn more about Equal Experts get in touch.