dynatrace

Managing Risk During Application Modernization Efforts

Using advanced performance monitoring to accelerate the refactoring of legacy apps to microservices

Table of contents

Executive Summary

Enterprises looking to leverage platform-as-a-service (PaaS) such as Pivotal Cloud Foundry and Red Hat OpenShift, typically start with two application environments:

1. New initiatives (greenfield) that can be built from the ground up using cloud native best practices
2. Existing application environments (brownfield) that they would like to modernize for the agility, scalability and performance benefits of cloud

Many of the existing workloads (brownfield) are critical applications to the business, generate substantial revenue, and are sensitive to disruption. While it would be ideal to completely re-architect these applications using cloud native principles, it is generally not practical or cost effective to do so.

This briefing provides practical advice and best practices on where and how to get started with the refactoring of a monolithic application. In combination, this helps IT teams accelerate the modernization process and reduces the risk of impacting customer events and business processes along the way.

Where to start?

Identifying candidates and components for refactoring

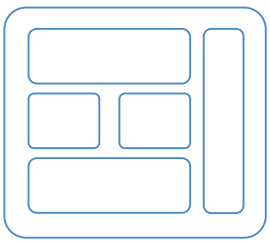
Before we begin, let's set the stage with a couple of assumptions:

- **You are starting with a well-structured monolith**
  Before you start the process of refactoring elements of your monolith, you will need to do some work to ensure that it is well-structured. If you are not sure if you are starting with a well-structured Monolith, learn more by taking a look at this video "Modular Monoliths" by Simon Brown (@simonbrown), an industry consultant in software architecture.

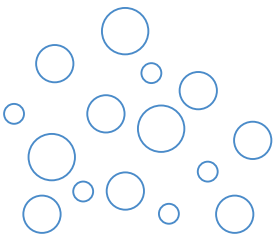
- **You are using DevOps practices and tools**
  To get the most out of cloud-native you need to be following DevOps best practices and using automation tooling for continuous integration (CI)/continuous delivery (CD). For more information on this, take a look at the eBook "Lessons Learned While Writing The DevOps Handbook" by authors: Gene Kim, Mark Tomlinson, Andi Grabner.

What is a monolith and how do I refactor it?

We tend to think of monoliths as being something bad in today's new world of distributed, loosely coupled application architectures. If you have a well-structured monolith, the only bad parts about it are that it is a single codebase and to scale it you must scale the entire thing. At its very essence, "From a software perspective, a monolith is a combination of business capabilities or bounded contexts all enmeshed with a core context. The process of transforming an app to a microservices based architecture entails separation of each of these bounded contexts into their own contexts."[1]



Monolithic/Layered



Micro Services

We tend to think of monoliths as being something bad...the only bad parts about it are that it is a single codebase and to scale it you must scale the entire thing

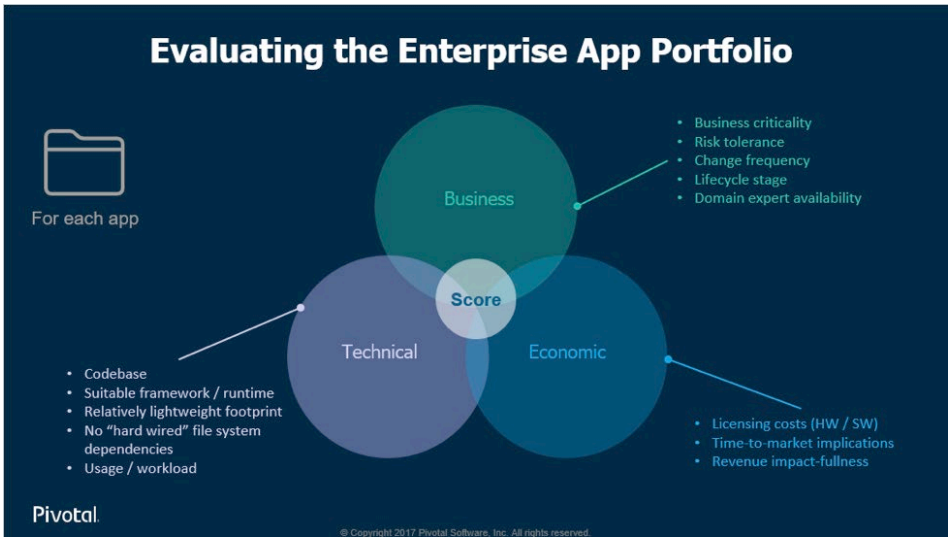
1. cloud.rohitkelapure.com/2016/05/application-transformation-pivotal-way.html

Section 1

Identifying Candidates for Refactoring

Deciding where to begin can be a daunting task. There are many things to consider. Some are business related, some technical related, and there always seems to be an economic component in the process.

Here is how Pivotal suggests you start identifying candidates for refactoring:



Source: Refactoring the Monolith: A Systematic Approach to Application Modernization

Once candidates are identified…"find the seams" in your monolith and start to refactor the code around those

Business considerations:

How critical is your custom application to the business? What level of risk can be taken with the app, and how frequently does it change? Are domain experts available?

Technical considerations:

Is it a suitable code base or is it riddled with twelve-factor violations? Does it use a suitable framework and runtime? What type of workload is it?

Economic considerations:

What amount of hardware and software investment does the app (or apps) merit? What sort of time window for a replatforming effort is appropriate given the expected outcome? What are the anticipated benefits or impact on revenue/cost savings?

Once candidates are identified, Pivotal recommends that you "find the seams" in your monolith and start to refactor the code around those. Seams should be thought of as "a particular business capability. Ideally it should

be something that must be isolated and adds business value."[2] Finding the seams can be difficult, and there are several ways you can approach it.

- Talk with the teams that maintain the application
- Talk to the users of the application
- Static code analysis with code coverage tools
- Event storming[3]
- <Enter your idea here>

Avoiding Analysis Paralysis

As you can see, there are many ways to try and identify where you should start. They can lead you in numerous directions and each has its pluses and minuses. The key is to start. As Simon Elisha from Pivotal said, "Getting overwhelmed by your legacy infrastructure, your existing estate, is often the biggest problem. There's so many, "But we can't because of this system or that system, etc." It's amazing when you start turning that thought process around, this is a cultural thing, to, "We can, and this is how we will do it. Or we'll try and see if it works."[4]

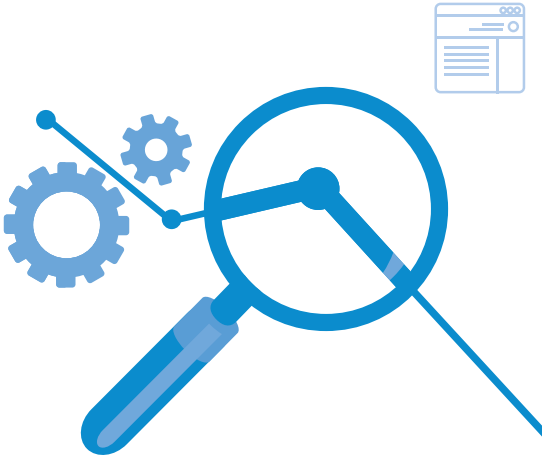
> "It's amazing when you start turning that thought process around, this is a cultural thing, to "we can, and this is how we will do it. Or we'll try and see if it works."
>
> *–Simon Elisha*
> *CTO, Pivotal*

Using a deep transaction monitoring solution can help you avoid "analysis paralysis" because it provides clear metrics to guide you to a decision. Fellow Dynatracer Mike Villiger (@mikevilliger) created a "Lenses" approach to identify potential functionality for refactoring:

Lens One: User Experience and Behavior Analytics

Real User Monitoring (RUM) can quickly expose candidates for refactoring based on activity involving those components. Looking at action frequency, patterns of interest emerge:

- **Highly accessed components** – These are critical components of your application and there may be great business value in carving them out. They may produce a higher payoff faster, but doing so can come at a higher risk to the business.
- **Less accessed components** – These aren't business critical, but may still be valuable in carving out and refactoring. The risk to the business is low and doing so allows you to do a proof of concept (ex. reCAPTCHA functionality).



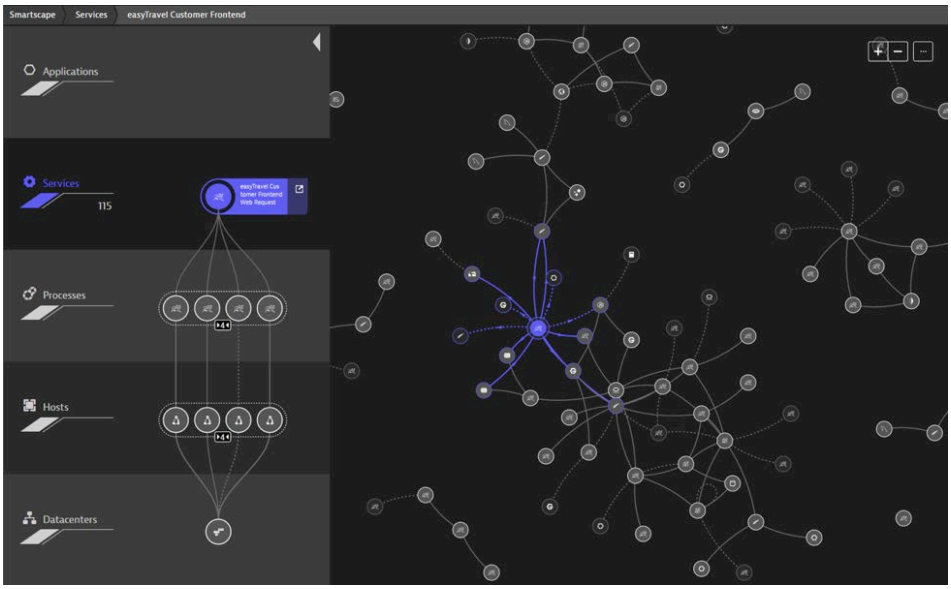
2. cloud.rohitkelapure.com/2016/05/application-transformation-pivotal-way.html
3. ziobrando.blogspot.com/2013/11/introducing-event-storming.html
4. content.pivotal.io/blog/decoding-the-monolith

- **Action duration** – Identify elements in the application that take a lot of time and contribute to a poor user experience. Identifying a poorly performing piece of functionality can be a great candidate that delivers high impact.
- **Compound metric** – We call it time consumed (Duration x Action Count). This exposes functionality that has the most impact on end user experience.

Lens Two: Topology & Dependency Discovery

Moving away from user experience, we can also look at how all the pieces of the application communicate with each other and the dependencies that exist currently.

- **Service Flow** – This shows you a user facing transaction and delivers it in an easy way to see how that transaction moves through your system. You can identify components that are heavily accessed and components that are lightly accessed. Again, based on the risk appetite, refactoring heavily used components will bring with them greater risk vs. less used components.
- **Topology Mapping** – Even though this is a monolith, you will rarely have a monolith that is truly operating in isolation. A topology mapping tool, like the Dynatrace Smartscape®, allows you to create a real-time map of all components and their dependencies. This may expose dependencies you didn't even know existed.



Dynatrace Smartscape

A topology mapping tool, like the Dynatrace Smartscape® allows you to create a real-time map of all components and their dependencies

Lens Three: Infrastructure Metrics

You may want to identify components that are heavy consumers of system resources and use that as a starting point. With APM, you can view resource consumption for different components:

> From the IaaS/Hypervisor

> To the Virtual Machine

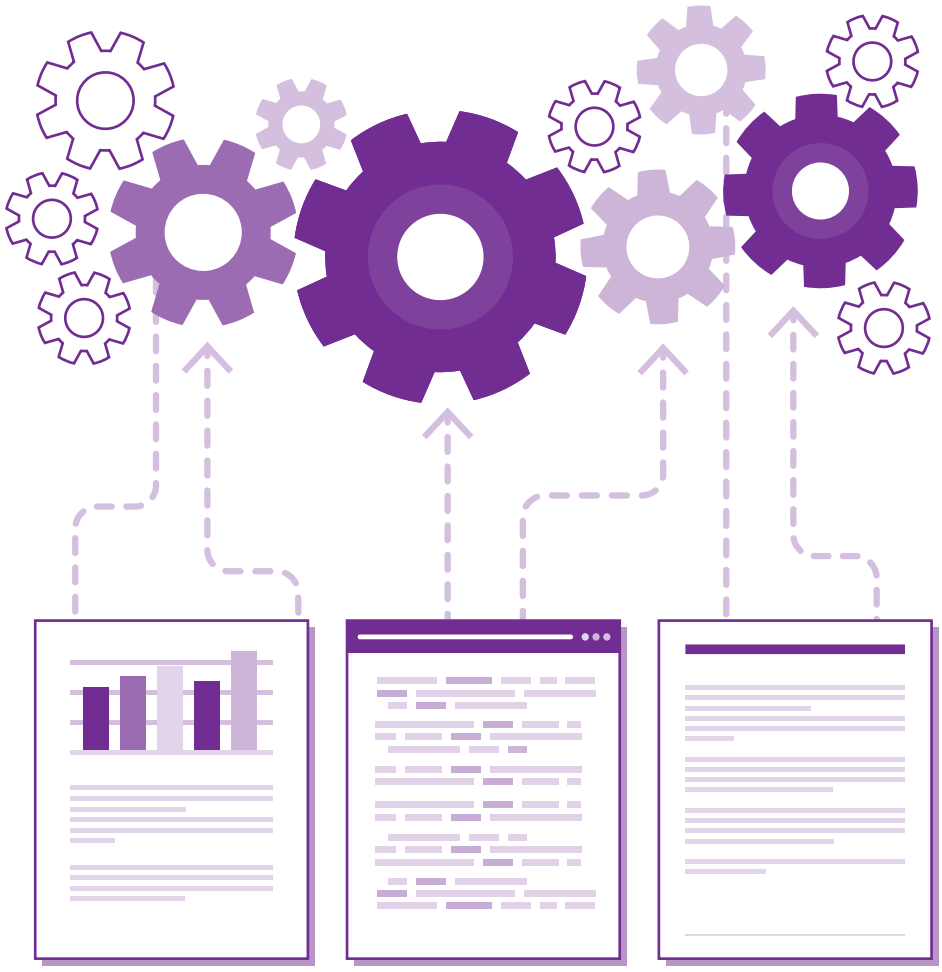
> To the process

> To the service

These lenses will help you identify candidates for replatforming, based on what the goals are of the organization and the impact a refactoring can have on performance.

Monitor the User Experience

As you refactor, the likelihood that you will be able to do this in isolation is probably low. Plus, it can be advantageous to let real users interact with your changes. As you move through the process, you will want to carefully monitor the impact of each change to ensure that the end user experience isn't impacted.

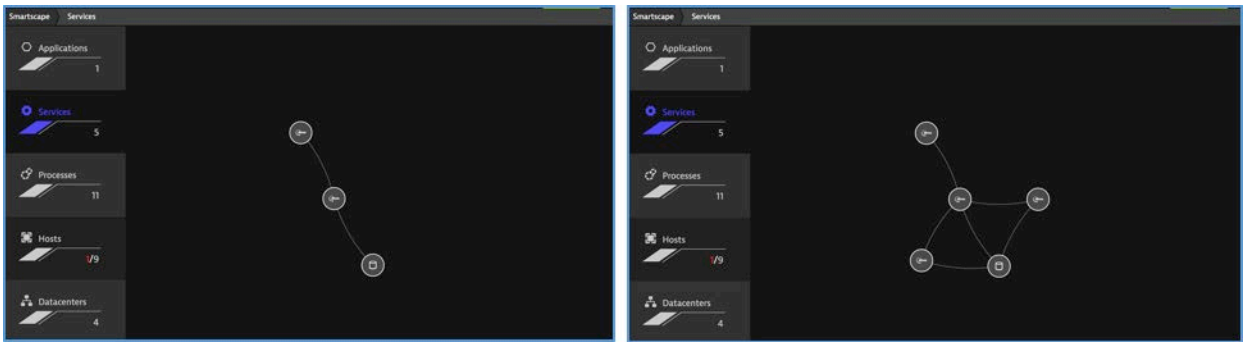


Section 2

Using Dynatrace to accelerate the move to microservices – Modeling & architecture validation

Understanding service interactions inside the well-structured monolith

Before you start to refactor your application, you should first model what those changes will look like. As an example, let's say you have identified the functionality for "payments" in your monolith. Within the monolith it could look like the image on the left side in the Dynatrace Smartscape. Dynatrace allows you to model the payment function as a microservice (the right-side image) and "virtually break the monolith." By doing this, it ensures that the architecture is validated and that tiers aren't inadvertently crossing boundaries.



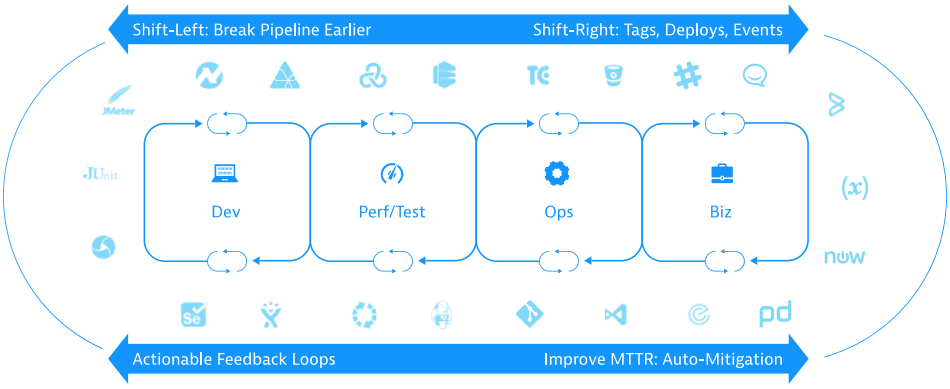
Modeling functionality as microservices

Additionally, you can use service flow information to continue to validate the architectural structure. You want to make sure you identify the bounded context that represents the way we are refactoring the monolith. Doing this lets you look at how a change to one part of the system will impact the rest of the application.

Dynatrace Service Flow

Section 3

Using Dynatrace to accelerate the move to microservices – leveraging AI-powered automation within DevOps processes



Dynatrace seamlessly integrates with your DevOps tools. Because of this integration, things like Tags and Environment Variables can be automatically pushed into Dynatrace from your DevOps tooling, and vice-versa, via our REST API and Command Line Interface (CLI). By integrating with Dynatrace, the Dynatrace AI engine can use these variables to help identify when a problem happens, what caused it to happen, and why.

To reap the benefits that automation can provide to your CI/CD pipeline, Dynatrace recommends a **Shift-Left** and **Shift-Right** approach. This will raise the bar on your DevOps strategy by increasing agility and velocity, and help you create maintain exceptional quality in your code.

- **Shifting Right** – Using information from your DevOps tooling (Tags, Environment Variables, etc.) into Dynatrace to help Dynatrace identify when changes occur and the impact they have on performance. When we Shift Right, we are using the information to identify problems faster and improve mean time to repair (MTTR).
- **Shifting Left** – Information from Dynatrace is pushed into your DevOps tooling to help you identify problem code faster. When we Shift Left, our goal is to break the pipeline earlier by identifying bad code as soon as possible and provide actionable feedback on how to fix it faster.

DevOps

As stated earlier, it is assumed that you are employing DevOps practices in your organization. DevOps has grown out of agile development and is a cultural practice. It emphasizes collaboration and orchestration across the entire organization, including business stakeholders, development, testing, deployment, and operations team; and allows organizations to quickly release new software to maximize business opportunities.

To learn more about DevOps, check out, Join the journey: Practical tips for scaling DevOps.

"DevOps is fluid, and avoids much of the traditional handoff friction and delays between product development and IT operations through greater collaboration, communication, and joint responsibility for the success of software delivery."[5]

– Pivotal

5. pivotal.io/devops

Shifting Right

Blue/Green Deployments, Canary Releases, Feature Flagging

> Dynatrace automatically identifies tags based on metadata on the host, process and service level.
  - Those tags can be used to identify new code deployments, such as blue/green deployments, and monitor them to see if the blue is performing as well/better than the green.

> Adjustments and rollback can be done automatically (see Auto & Self-Healing below).

> A ticket can be created/updated automatically with the cause of the problem, what was done to mitigate it and whether the changes solved the problem.

Auto & Self-Healing – While this is more valuable as an ongoing production capability, it does prove useful if you are refactoring a live application and pushing the newly created microservices into production.

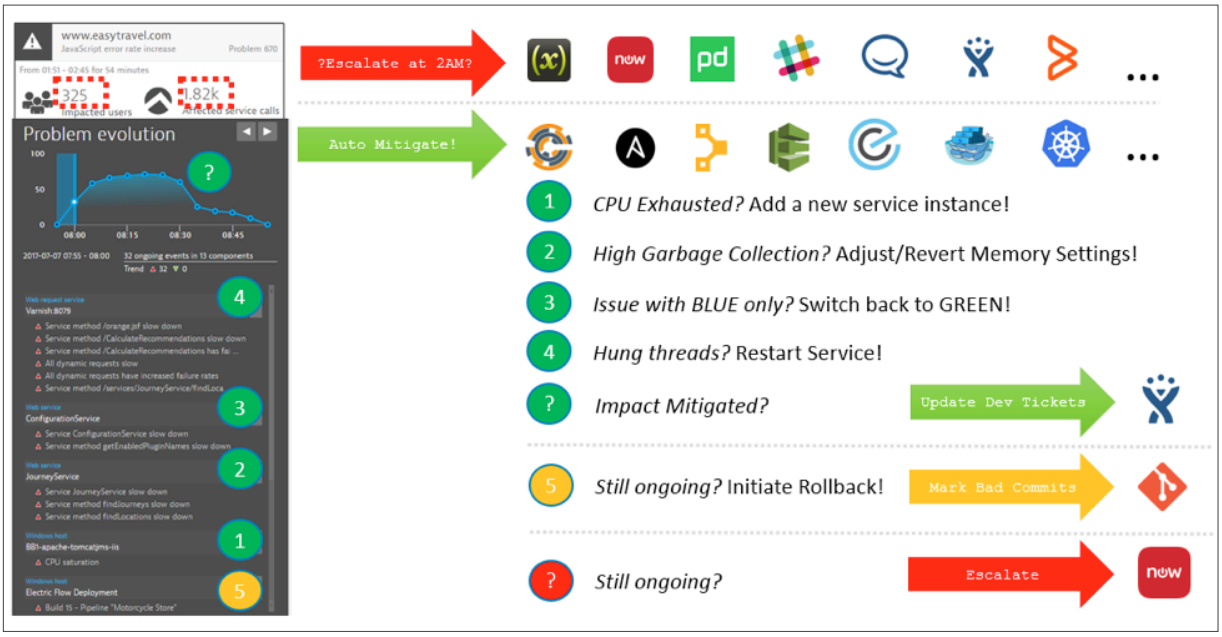
> Using Tags and identifying specific responses to identified problems, Dynatrace can automatically alert your CI/CD tooling to make adjustments in your environment, from a simple memory change all the way to a full roll-back.

> Information from Dynatrace can automatically be populated into the ticketing system used by developers to help them identify what caused the problems.

The benefits of automation across the application delivery pipeline

"In our research we found that organizations which have adopted automation across the complete lifecycle including continuous integration, continuous delivery and production deployment have experienced more velocity and quality. Purposeful end-to-end automation is foundational to success especially as DevOps initiatives scale."

- Rob Stroud
Principal Analyst, Forrester
[October 2017]



Problem AI: Incident and Self-Healing

Shifting Left

Enable Fast Performance Feedback along the DevOps Pipeline - Dynatrace can help you detect all potential performance, scalability and architectural problems early, with details down to the root cause.

> Utilize Dynatrace time-series API to fetch detailed data related to both response time and resource consumption metrics across test periods for external comparison.

> Poll Dynatrace problems API to determine environment state before attempting an automated deployment.

> Feed Dynatrace data into Agile story tracking tools to document success (or failure) of a new feature; both for business focused metrics like action counts and conversation rates as well as technical data like response times and resource consumption.

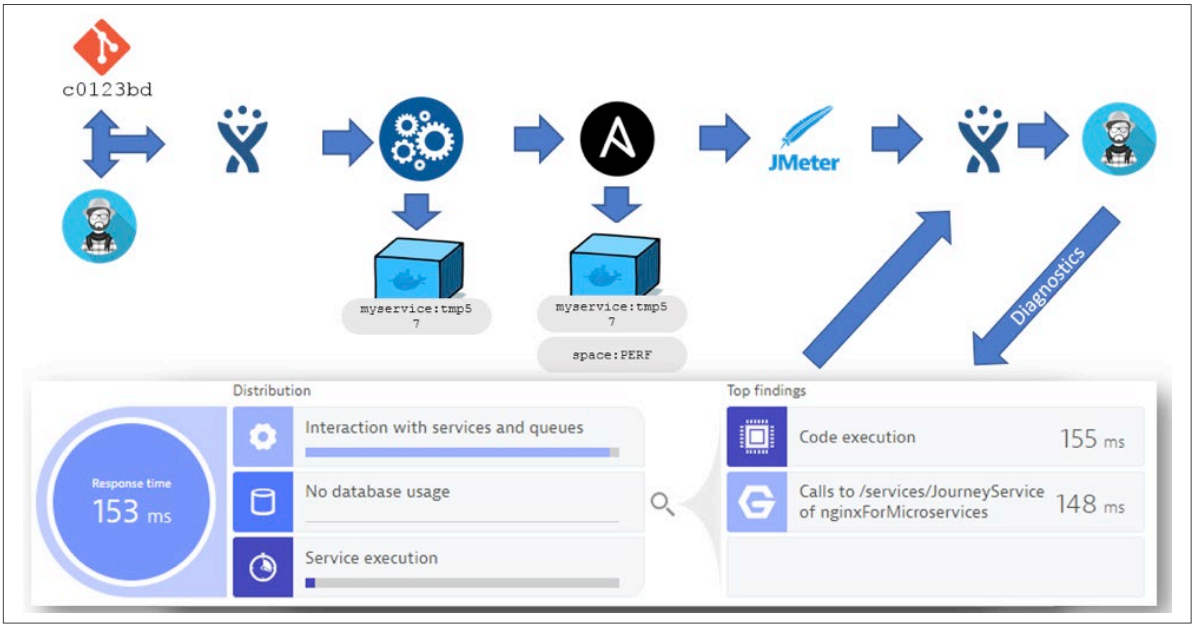
Self-Service Performance Testing – Working with your current DevOps tooling, Dynatrace allows developers to do performance testing on new code without the need for performance engineers.

> A developer can create a ticket and kick off a build into the performance testing environment (works off tags).

> Results from Dynatrace and load testing software automatically gets added to the ticket.

> Originating developer gets performance results and access to diagnostic details via a link in the ticket.

> The performance testing function is optimized and it allows you to scale your development team's productivity.



Self Service Performance Testing with Dynatrace

These are a few high-level examples of ways you can use Dynatrace to accelerate the development of microservices as you refactor your monolithic application. For more in-depth information on each of these examples, please refer to the links in the Additional Resources section.

When is refactoring complete?

To quote Matt Stine from the O'Reilly eBook, "Migrating to Cloud-Native Application Architectures":

How do we know when we are finished? There are basically two end states:

1. The monolith has been completely strangled to death. All bounded contexts have been extracted into microservices. The final step is to identify opportunities to eliminate anti-corruption layers that are no longer necessary.

2. The monolith has been strangled to a point where the cost of additional service extraction exceeds the return on the necessary development efforts. Some portions of the monolith may be fairly stable—we haven't changed them in years and they're doing their jobs. There may not be much value in moving these portions around, and the cost of maintaining the necessary anticorruption layers to integrate with them may be low enough that we can take t on long-term.

What is strangling the monolith?

"The most important reason to consider a strangler application over a cut-over rewrite is reduced risk. A strangler can give value steadily and the frequent releases allow you to monitor its progress more carefully. ... Since you can use shorter release cycles with a strangler you can avoid a lot of the unnecessary features that cut over rewrites often generate."

- Martin Fowler
StranglerApplication

Additional Resources

BLOGS

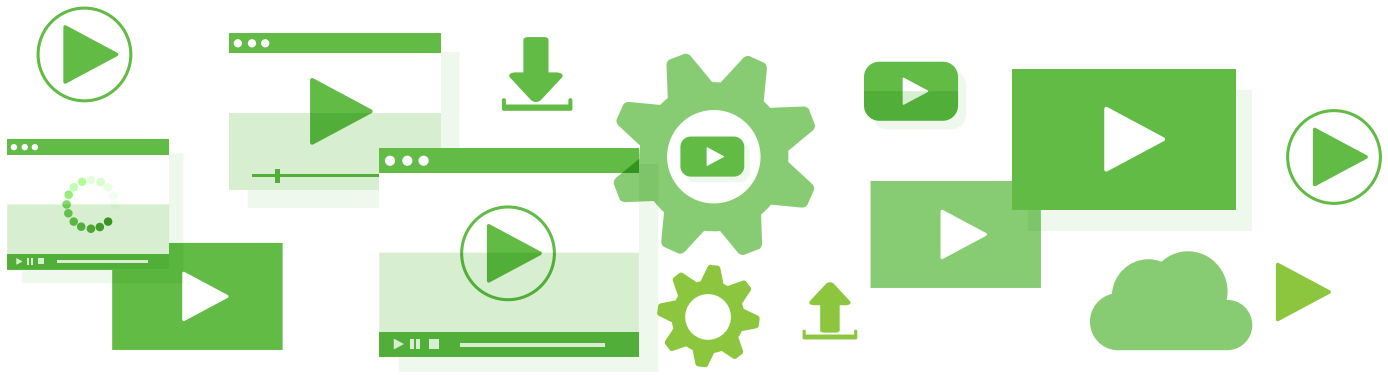
Auto-Mitigation with Dynatrace AI – or shall we call it Self-Healing?
Unbreakable DevOps Pipeline: Shift-Left, Shift-Right & Self-Healing
Application Transformation – The Pivotal Way

VIDEO

Building an Unbreakable Delivery Pipeline: Shift-Left, Shift-Right & Self-Healing

WEB

Scaling DevOps to deliver better software faster



Learn more at dynatrace.com

Dynatrace is the innovator behind the industry's premier Digital Performance Platform, making real-time information about digital performance visible and actionable for everyone across business and IT. We help customers of all sizes see their applications and digital channels through the lens of their end users. Over 8,000 organizations use these insights to master complexity, gain operational agility and grow revenue by delivering amazing customer experiences.

04.12.18 2252_ManagingRiskAppMod_jg