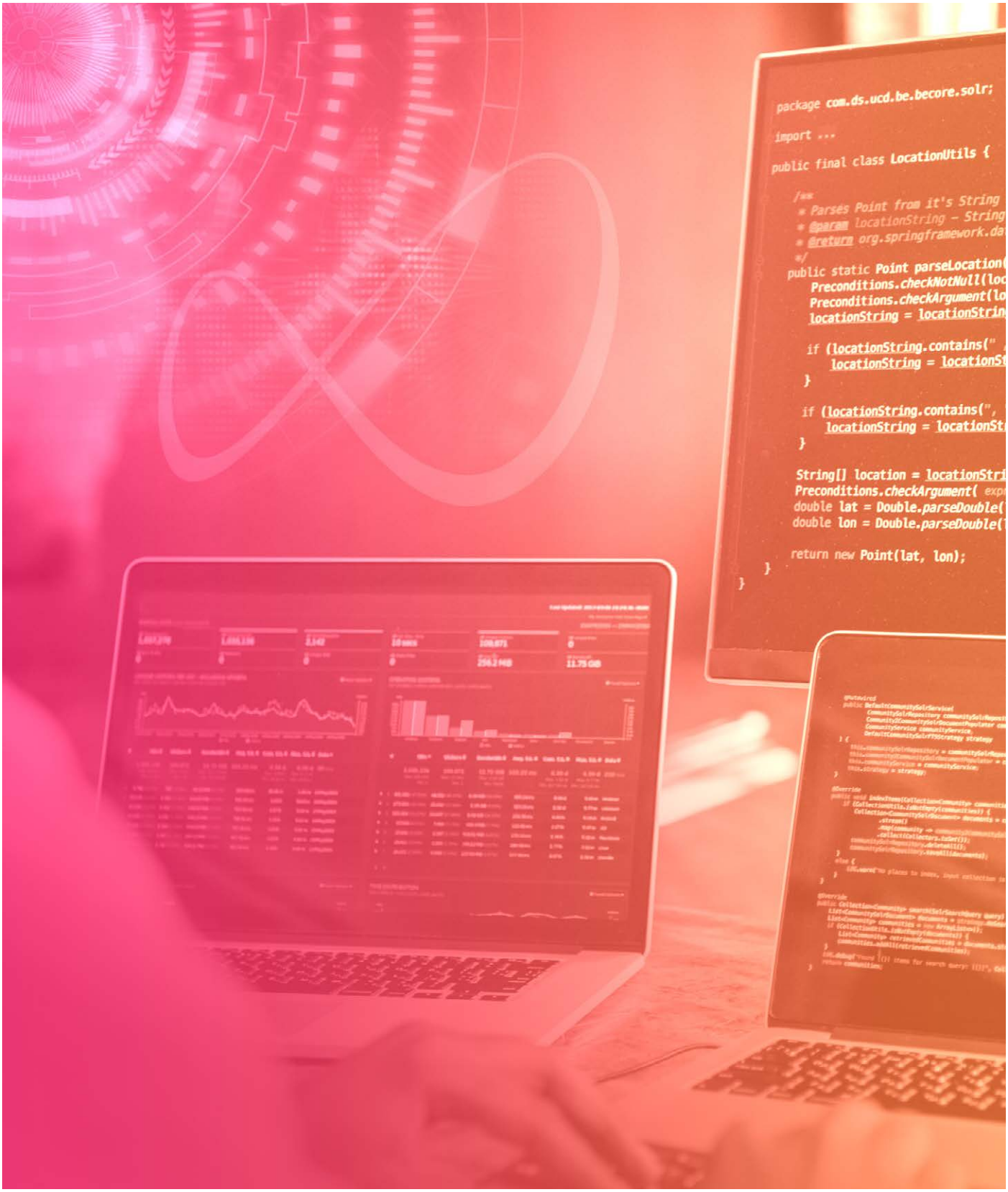
WHITE PAPER

Mphasis stelligent
The Next Applied

Intro to DevOps Automation on AWS



CONTENTS

1.
Intro to DevOps Automation On AWS

Our guiding principle with customers is to automate all processes until they can be run from the click of a button and from a single command. The reason we do this is to reduce human bottlenecks. Anytime a human needs to run a process, it delays feedback and its correction. By eliminating these bottlenecks, teams can release software systems whenever they choose to do so. As illustrated in Figure 1, we apply a six-step heuristic to all of these processes. The six steps are: document, test, code, version, continuous, and monitor.

This is another way of saying that we use good software engineering practices when it comes to infrastructure and that infrastructure code works as part of a larger ecosystem of techniques and tools. Additional considerations include microservices and self-service mechanisms — which are approaches and principles around the architecture in how you deliver your software systems.

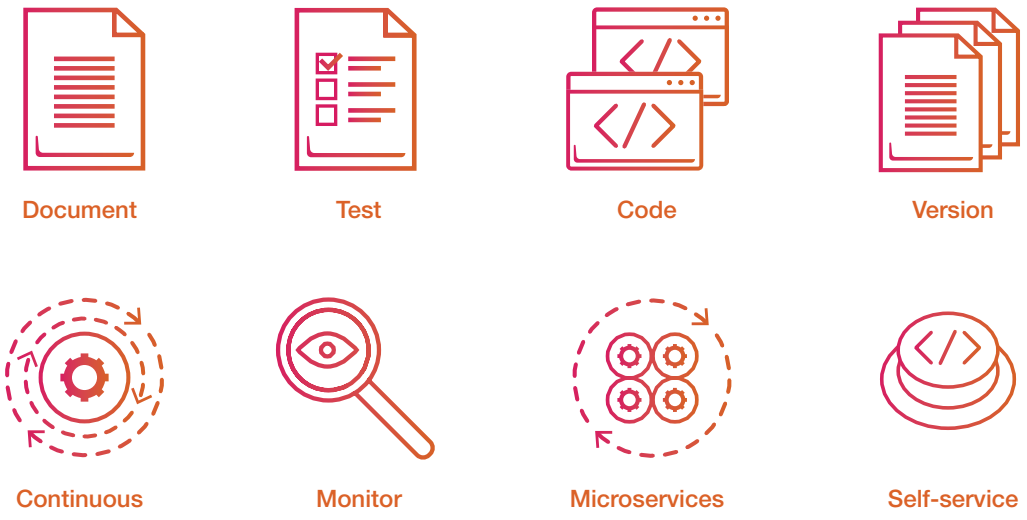


Figure 1: Introduction

2.
The DevOps Ideal

Any authorized team member can have an idea in the morning and have it confidently deployed to production in the afternoon of the same day.

Let's dissect this statement a bit...

Any authorized team member - This means any cross-functional team member (e.g. application developer, database, UI/UX, QA/Testing, or Infrastructure) who is trusted by the team.

An idea - This might be a feature, a fix or whatever and this might be any change: application code, configuration, infrastructure, or data. In other words, anything that makes up the software system.

Confidently deployed - This means it's performed securely, with reduced risk, and using a single path to production. You're confident it's going to go through the same approved process every time on its way to production.

Afternoon - This means there's a short cycle time as it goes through a single path to production. You also know it's not a toy as it's going to production.

3.
What is DevOps?

DevOps is a portmanteau of the words representing "Development" and "Operations" teams, but it's really more about representing the entire value stream.

What you see in Figure 2 is similar to what AWS shares in some of its DevOps talks — by relating DevOps to something you're likely familiar with: the software development lifecycle.

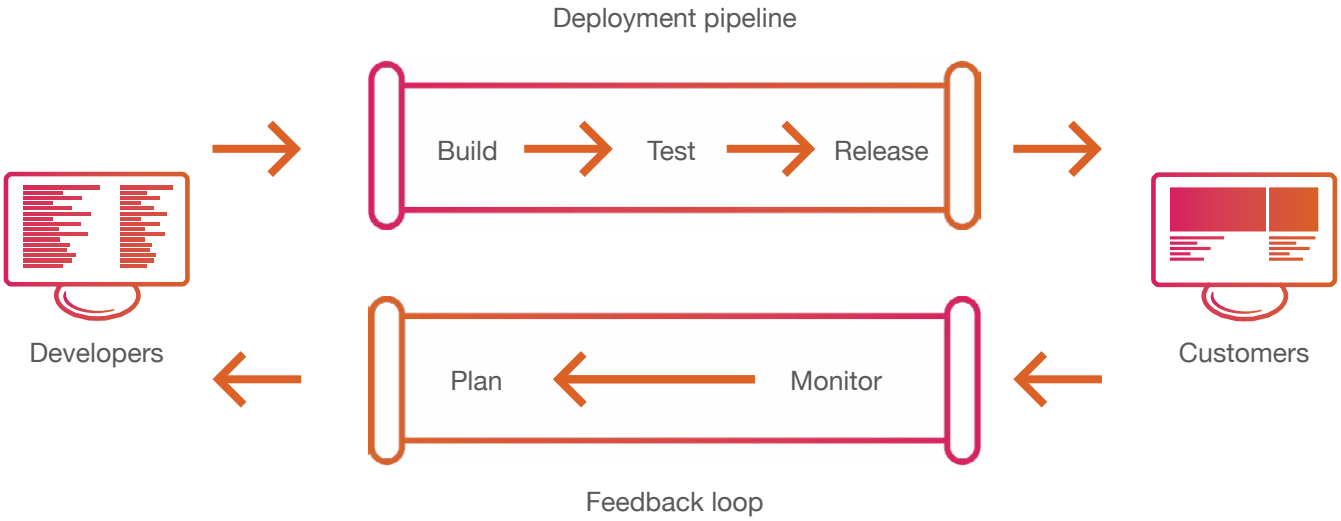


Figure 2: What is DevOps?

This might be for a web application or for a service. On the one side, you have customers and the other, developers. A developer comes up with an idea for a new feature, implements it and then puts it through a process of building, testing, and going through a release process until it is delivered to production where your customers actually start using it. It's only once it gets into the hands of your customers that you start to learn from it. You can get usage data, get direct feedback from customers, or start to make informed decisions on what to work on next. Based on this, you might decide to update or improve the feature, or even develop a new feature — and then the feedback loop starts again.

There are two key points to consider:

- The faster you're able to get through this loop determines how responsive you can be to customers and how innovative you are
- From your customer's perspective, you're only delivering value when you're spending time on developing new features

Therefore, you want to maximize the time you are spending on developing new features and minimize the time you are spending on the process for building, testing, and releasing software systems.

So, it's really these two things that makeup DevOps. As a result, any efficiency you can drive into the middle to increase these feedback loops is DevOps. In addition, though it's to be expected, it often confuses people because this could mean changes to the culture, organization, process, or tooling. Improving anything in this feedback loop is what DevOps is all about.

4.
A DevOps Definition

From our perspective, the definition of DevOps is:

Confidently speed up feedback loops through organizational, cultural, process, and tooling changes as a means to increase experimentation, increase confidence, reduce risk, and reduce costs.

While the other facets are important, the focus of this whitepaper will be on tooling and automation.

5.
DevOps Automation

In automating the entire software delivery lifecycle, teams can deliver software at the click of a button. Every part of the software delivery process is automated — from commit to production. This includes the application code, configuration, infrastructure and data — really everything.

One way to visualize this is to imagine a factory assembly line where cross-functional engineers work on various parts of the software as it moves in its lifecycle.

In the past, teams might only deliver software every few months or so — because of manual, error-prone processes. Often pieces of the delivery package go back and forth between teams separated by silos. This is a

long, drawn-out and expensive exercise with the final package assembled at the last minute with teams vowing never to repeat it again.

With DevOps Automation, software can be delivered several times a day, once a week or as often as you want - it's always in a releasable state.

When compared to traditional delivery methods, this offers two key advantages. The first is that the software is always ready to release meaning you do not stop and make a special effort to release the software. The second is that there aren't any walls between the teams. Instead, there are autonomous teams made up of developers, testers, infrastructure, etc. all contributing to a single path to production.

We're not just talking about application developer changes here. No matter if it's a change to the infrastructure, data or whatever; the entire package gets built, tested, analyzed and deployed. When it passes all these checks, it becomes a release candidate for potential release to users, if the business chooses to do so.

Now, when a problem is discovered, quick feedback notifies team members that something's wrong that needs to be fixed and the assembly line stops. This is when team members make it a priority to fix the error and commit it so that it moves along in the path to production.

So, in summary, it's concerned with all parts of the software system and how all team members work as part of this single path to production. This way, software gets to users quicker and in a more predictable manner.

To see a video that visualizes this, go to https://www.youtube.com/watch?v=SIaVsG7m8n4

6.
Practice 1: Document

If you are working on a new project, you might first write documentation for manually provisioning software system resources in such a way where they can be automated later.

If it's an existing project, you might need to rewrite the documentation so that it can be automated. If you're already familiar with a set of tools and technology you're configuring, you might jump right into writing automated tests. Once the tests and code are committed to version-control for a particular process, we usually don't have need for this documentation, so we often remove it.

There's also a purpose to documentation outside of provisioning and configuring resources. Most ideally, you want the ability to codify the formatting as well. This way you can version it more easily. An example of defining documentation that's code is using Markdown.

You might define your documentation and diagrams using tools like CloudFormation Designer, Confluence, Markdown, and Cloudcraft.

7.
Practice 2: Test

After writing the initial documentation, you might write automated infrastructure and deployment tests based on the documentation and/or as a way of describing the system specifications for infrastructure, environments and deployments. We tend to write infrastructure/deployment tests based on risk not on achieving "100% code coverage" (which is a fantastically elusive concept in infrastructures anyway) and as a way to describe what the system should do. An example of an infrastructure

test platform we use is ServerSpec, but there are also some interesting ways to instrument monitoring to perform tests and remediate in production too.

Some example testing and static analysis tools you might use include:

- Running tests via AWS Lambda and AWS CodeBuild
- Gauntlt
- config-rule-status
- inspector-status
- ServerSpec

8.
Practice 3: Code

Treat everything as code: the application code, the configuration, the infrastructure and the data. Whether it's the infrastructure, the build, the deployment or the application or service, treat everything as a first-class software artifact.

Some example infrastructure as code tools include:

- AWS CloudFormation
- Docker
- Chef

For example, in AWS CloudFormation you define everything in a domain-specific language via JSON or YAML and it's all in code. Therefore, you can use something like AWS CloudFormation to provision your servers, network, storage, and databases - all of this in code.

9.
Practice 4: Version

Next, ensure all assets are versioned - application and test code, configuration, infrastructure and data in version-control systems like Git. This also includes build and deployment scripts, and deployment pipelines as well - all of it can be defined in code.

Common examples include:

- AWS CodeCommit
- GitHub
- Atlassian Bitbucket

10.
Practice 5: Continuous

Another pattern that a high-performing organization uses is something known as a deployment pipeline or a “single path to production”. This pipeline is a fully automated implementation of your build, deploy, test and release processes. This automation is part of a software delivery system composed of stages and actions. Each stage and action provides actionable feedback to the team and to the individuals who recently committed the code. A Continuous Delivery service is established to poll the version-control repository. When it discovers these changes, it kicks off an instance of this pipeline. Therefore, with every commit, team members can check the status of the pipeline to see if the recent changes were successful. If any of these actions fail, the pipeline stops and there should be no more commits to the repository until the fixes are applied. This approach works best when committing changes in small batches and deploying those changes to production in small batches. With this pipeline, you can potentially deploy/release software to users several times a day or less often depending on your release cadence. This approach gives you the flexibility to deploy or release changes

whenever you choose to do so as the act of deploying/releasing is essentially a “nonevent.”

Common examples include:

- AWS CodePipeline
- Jenkins 2
- CircleCI
- Atlassian Bamboo

11.
Practice 6: Monitor

Once you have documented, written some tests, codified it, versioned the code, and made it continuous, you can monitor all activity so that you are both passively and actively informed of behavior that might require remediation. The exciting thing is that, on AWS, you can automate the provisioning of all your monitoring resources in code and you can automatically perform remediation based on rules that you establish within your account. Some of the different ways you can monitor are:

- Amazon CloudWatch - “Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications you run on AWS. You can use Amazon CloudWatch to track metrics.
- AWS CloudTrail - “AWS CloudTrail is a service that enables governance, compliance, operational auditing, and risk auditing of your AWS account.”ii
- AWS Config - “AWS Config is a service that enables you to assess, audit, and evaluate the configurations of your AWS resources.”iii But what’s most exciting is that you can use AWS Config “to codify your compliance with custom rules in AWS Lambda that define your internal best practices and guidelines for resource configurations. Using AWS Config, you can automate assessment of your resource configurations

and resource changes to ensure continuous compliance and self-governance across your AWS infrastructure" — in other words, automated compliance and remediation (if necessary).

- https://aws.amazon.com/cloudwatch/
- ii https://aws.amazon.com/cloudtrail/
- iii https://aws.amazon.com/config/

12.
Practice 7: Microservices

Microservices are small, independently deployable software services that communicate via contracts between other services (i.e. APIs). One page might have hundreds of services each run by small autonomous teams. DevOps Automation does not require microservices - but it works much better with microservices. On the other hand, microservices essentially cannot function well without DevOps Automation. Microservices is its own discipline and covered in several books; see the Additional Resources section for more information.

13.
Practice 8: Self-Service

The goal of self-service deployments is to give any authorized team member the ability to experiment, investigate, and make destructive changes that do not affect the canonical service that's been committed to the version-control repository.

At any time, an authorized user on a team should be able to run a self-service deployment.

This is known as a pull-based mechanism as the team member does not need to wait for someone else to push the deployment to them. The team member should have some way to indicate the version of the service that they would like to deploy into an environment. This version should be based on changes that have gone through

a pre-approved set of stages by the team so that there is a level of confidence of what has been automatically tested.

There are many ways to employ self-service deployments on AWS, but the AWS Service Catalog is one such service that can enable them in a structured manner.

For example, a developer might want to manually make changes to an environment as a way of learning how to automate the various components in the environment.

You will likely want to employ a self-destruct mechanism into the environment so that there aren't too many unused environments laying around after being deployed.

14. Getting Started

One of the worst things you can do is attempt to fix everything all at once or even feel like you have to get it perfect the first time. Every team and organization is different so there's going to be some trial and error. Start small and iterate.

This section describes how you might assess and plan for your DevOps transformation when considering automation.

Assess

We often work with customers to formalize the process of preparing for, and kicking off such an endeavor. We call this service a "Stelligent Roadmap." The service typically takes about two weeks and is highlighted by a very open and earnest examination of a customer's "automation readiness state," which results in an objective "Scorecard."

A typical engagement consists of 50 questions that we ask teams and then verify through multiple sources. We use this to develop a detailed report of recommendations and scores across various dimensions. These recommendations are then used in developing a plan of action for transformation.

Our assessment includes organizational, process, and cultural aspects as well as tooling and automation. Here is a sample Scorecard (from the Stelligent Roadmap service).

If you are doing this on your own, some areas you might consider include:

- Active Configuration Monitoring & Automation
- Application Monitoring & Automation
- App/Service Deployment
- Auditing
- Binary Artifact Storage
- Blue Green Deployments
- Build Automation
- Build Distribution Storage
- Code Quality Analysis
- Configuration Secrets Management
- Container Architecture
- Cost Monitoring
- Database Provisioning & Updates
- Decommissioning Resources & Automation
- Deployment Pipeline Architecture
- Disaster Recovery & Automation
- Deployment Documentation
- Event-Driven Monitoring
- Feedback Mechanisms
- Infrastructure and Deployment Diagrams
- Local Development Environment
- Logging & Automation
- Manual Actions as part of a fully-automated workflow
- Infrastructure Automation: Network, Node, Deployment Pipeline
- Infrastructure Code Pipeline

- Metrics: Pipeline Wait Times, Build Frequency, Cycle Time, Deployment Frequency, Mean Time to Detect, Mean Time to Recover
- Production Deployment
- Patterns: Autonomous Teams, Code Commit Frequency, Immutable Infrastructure, Self-Service Deployment, Stop the Line
- Security Group or Firewall Automation
- Self-Healing Automation
- Storage Automation
- Technical Onboarding
- Version Control Usage
- Automated Testing: Acceptance & Functional, Unit, Infrastructure & Deployment, Integration, Load & Performance, Penetration, Chaos (Resiliency), Compliance

Plan

Once we have assessed a team's capability to deliver software, we also collaborate with customers to envision the desired automation state, and provide them a "Roadmap" to move from their current to desired state. Thinking and speaking in agile terms, we actually populate an initial backlog with epic stories that represent a prioritized set of things that need to be done to achieve the desired state.

Here is a sample Roadmap (sample from Stelligent Roadmap service).

We have grouped the recommendations into phases that are intended estimate when tasks should be acted upon.

Phase 1 - Orchestrate Entire Software Delivery Workflow

Phase 2 - Improve Utilization & Costs

Phase 3 - Improve Security Posture

Epic	Description	Priority	LOE	Phase[1]
Deployment Pipeline Stages - Manual	Create an end-to-end deployment pipeline with manual approvals for manual actions and automated actions for existing automated actions	1-High	3-Low	1
Deployment Pipeline Stages - Automated	Commit, Acceptance, Exploratory, Preprod, Prod stages exist and are chained together	1-High	1-High	1
Version Control Usage	The entire software system is versioned (application code, configuration, tests, infrastructure and data) along with inary libraries used by the application in a single version control system	1-High	3-Low	1
Support Infrastructure Reproducibility	Each piece of the Support Infrastructure (NAT, Bastion, Binary repository, Continuous Integration server) is completely scripted and built by a CloudFormation template. Each piece of Support Infrasctructure has its own CloudFormation template with its own lifecycle	3-Low	2-Medium	1

Figure 3: Roadmap - part of Stelligent Roadmap service

15. Summary

In this whitepaper, you learned how you can codify all the things including documentation, tests, infrastructure and deployment pipelines. You also learned how to make all version-control commits go through a process of building, deploying, testing, and releasing software systems that are both actively and passively monitored via fully configurable managed services via AWS.

16.
Additional Resources

Here are some additional resources. The first link contains links to a myriad of popular blog posts, books, talks, videos, open source tools, and articles that people at Stelligent have published over the years.

https://stelligent.com/2017/01/03/sharing-for-the-people-stelligentsia-publications/

- https://github.com/stelligent
- https://github.com/stelligent/cloudformation_templates
- https://github.com/stelligent/cfn_nag
- https://github.com/stelligent/mu
- https://github.com/stelligent/config-rule-status
- Stelligent Roadmap: https://stelligent.com/services/preparation/
- https://stelligent.com/blog/
- DevOps on AWS Radio: https://stelligent.com/category/podcasts/
- https://twitter.com/Stelligent
- https://dzone.com/refcardz/continuous-delivery/patterns
- https://aws.amazon.com/devops/continuous-delivery/
- Building Microservices: Designing Fine-Grained Systems (Newman)
- Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization (Fowler)

ABOUT MPHASIS STELLIGENT

Mphasis Stelligent provides DevOps automation professional services on AWS, enabling engineering teams to focus on creating software users love. Our goal is to work closely with customers to develop fundamentally secure infrastructure automation code, deployment pipelines, and feedback mechanisms for faster, more consistent software and infrastructure deployments.

For more information, contact us at: info@stelligent.com

11710 Plaza America Drive
Suite 2000
Reston, VA 20190-4743
Tel.: +1 888 924 4539

Mphasis stelligent™
The Next Applied