2017

The Power Of DevOps Assembly Lines

shippable

Introduction

Software delivery has evolved in the past decade as organizations adopt a "DevOps mindset." As a result, development, test and operations teams have begun automating their tasks, and several tools vendors have emerged to support these activities. However, using a diverse and disconnected toolchain has now created "islands of automation" that are optimized for each silo, but do not provide visibility, auditability and traceability across the entire workflow. Many teams try to compensate for this through inefficient methods like meetings, spreadsheets or Slack conversations. Alternatively, they build homegrown, point-to-point workflows between these islands that are fragile and difficult to scale.

The next wave of DevOps is about creating assembly lines that connect these islands and bring the discipline of hyper-efficient manufacturing to software delivery. This whitepaper examines this evolution and explains why the Assembly Lines approach will help you best achieve the goals of DevOps and Continuous Deployment.

What is DevOps?

The word "DevOps" means different things to different people. There is some agreement about what it does for you: DevOps is a set of principles that help you remove inefficiencies in your software delivery workflow and release software rapidly, frequently, and more reliably. The ultimate holy grail is Continuous Deployment, when every change goes from source control to production in a fully automated manner with no human intervention, i.e. you can do zero-touch deployments.

However, the ***"how"*** of DevOps is still confusing. Some see it as a purely cultural transition where Development and Operations teams start collaborating more closely and exchanging information. Others focus on the automation aspect, which itself comes in many flavors and sizes. To add to the uncertainty, a plethora of terms are thrown around interchangeably, creating confusion and complexity, like Continuous Integration (CI), Continuous Delivery, Pipelines, Application Release Automation (ARA), and Continuous Deployment (CD). It is often difficult to figure out where each of these start and where they end. We have listed these in the Appendix, so you get a clear definition of what each term really means.

The State of DevOps today

As DevOps has gained traction in the last few years, it has greatly changed the mindset in many organizations and has underlined the need to be agile and to ship code faster. As a result, development, test and operations teams have begun automating their tasks, and several tools vendors have emerged to support these activities.

Most of the tools available today help automate specific sections of the software delivery workflow. Each of these sections is usually referred to as a 'Pipeline'. A **Pipeline** is a point-to-point sequence of instructions or activities that has a singular goal, such as deploying a commit to an endpoint, or pushing a Docker image to a hub. Each team usually has one or more pipelines that help automate their specific activities. Some of these are described below.

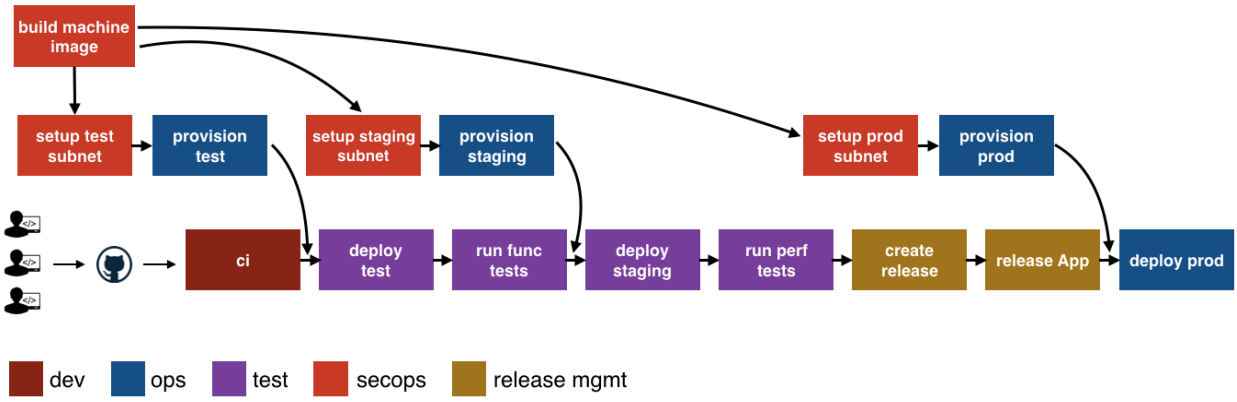


Figure 1: Pipelines required to ship one application or service

Continuous Integration: For every change, source code is built, unit-tested, and packaged into a Docker image, or tarfile, or JAR/WAR file. The package is pushed with an immutable tag to a hub or artifact repository.

Deployment: Every deployment to an environment needs a pipeline. For example, deploying a new version of the application package to the Test environment is a pipeline. So is deploying to Production after Staging.

Executing tests: Different test suites are run on different environments, such as Functional, Performance, Smoke test suites.

Software Defined Networking (SDN): In today's cloud world, a significant portion of networking, IP addressing and subnets, load-balancers and firewall rules are all authored "as-code". Automating these activities so that dependent applications can programmatically access critical networking configurations is critical to CD.

Machine Image Preparation: Virtual Machines (VM) are provisioned from corporate images with all the necessary packages, monitoring, and tooling built-in. The preparation of these images has to be automated and available for rapid and consistent provisioning of environments, avoiding environment drifts.

Environment provisioning: Machines across all application environments like Dev, Test, and Production, needs to be automatically provisioned.

Operating System Patching: Security patches are regularly released by software vendors. In a large organization, applying these unplanned time-sensitive patches across all machines is very disruptive and leads to a lot of frustration. Automating the rollout of patches across data centers is critical to reduce this disruption.

Backup and Disaster Recovery: These tasks need to be executed periodically across all production systems. The ability to integrate these backup locations into provisioning pre-production environments reduces friction involved in certifying a release.

Most organizations have seen some efficiency gains with a combination of pipeline automation and cultural changes. However, most DevOps efforts plateau after a point and it is difficult to measure success with a concrete metric. This is due to three major challenges:

- The tools available today lead to islands of automation with manual processes or point-to-point integrations between each island,
- Application architecture is evolving rapidly, and the current linear approach to software delivery does not handle the complexity of Microservices or Serverless architectures,
- Every organization has a heterogeneous application portfolio with no consistency in how they are managed or shipped.

Let us look at these challenges in greater detail.

Challenge #1: Islands of Automation

Even though many tool vendors have emerged, they are focused on helping automate pipeline(s) for a specific team, such as Development, Test, or Operations. As a result, teams are

forced to choose multiple disconnected tools across pipelines, and automating pipelines in isolation is creating disconnected **"Islands of Automation"**. To make things worse, every vendor makes their tool sound like a silver bullet, but once you do a deeper dive, you discover that the tool only gets you part of the way towards DevOps and CD.

The main problems with islands of automation are:

- There is no visibility across the toolchain, so finding bottlenecks is time consuming and challenging
- The islands aren't connected, so there is need for manual processes at the edge of every silo

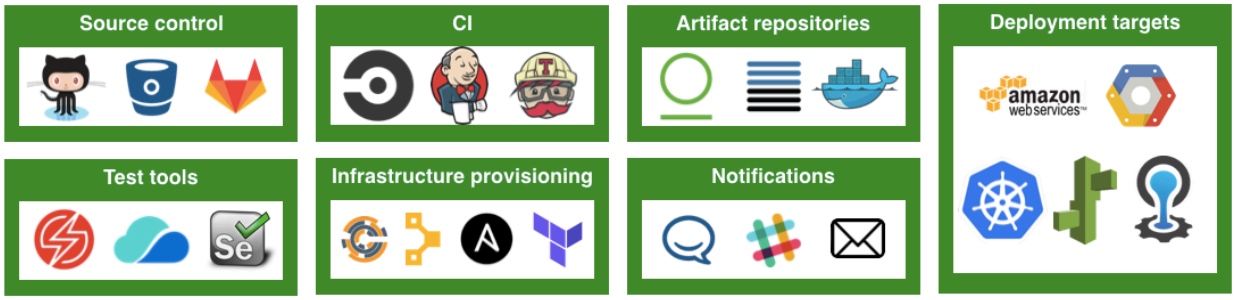


Figure 2: DevOps islands of automation

Many teams try to compensate for this through inefficient methods like meetings, spreadsheets or Slack conversations. Alternatively, they build homegrown, point-to-point workflows between these islands that are fragile and time-consuming to develop and maintain. Neither of these approaches is scalable and cause additional bottlenecks for shipping software.

Unless these islands of automation are connected together and act as one continuous workflow, you cannot achieve true Continuous Deployment, or even Continuous Delivery.

Challenge #2: Rapidly evolving application architecture

The second big challenge is that the "need for speed" is causing application architecture to evolve rapidly, from **N-tier** to **Service Oriented Architecture (SOA)** to **Microservices (MS).**

With Microservices architecture, applications are broken down into very small fragments, creating a complex web of dependencies within different fragments of the application. The number of pipelines required to ship these applications explode, and the linear limitation of Application Release Automation (ARA) is not enough to manage the dependencies. Manual activities at the edge of every silo also slow down the process of shipping microservices much more than they do with SOA.

It is essential to explore an approach that supports non-linear, interconnected workflows to address this challenge.

Challenge #3: Heterogeneous applications and tools

Beyond just application architecture, the overall technology landscape is also evolving very rapidly. In the last two decades, physical servers were replaced by virtual machines and virtual machines are now being replaced by Docker containers. Programming languages and tools are also changing, from C++ and C, to C# and Java, to today's popular languages like Node.js and Go.

This rapid evolution has led to every organization having a mix of heterogeneous applications. For example, you might have a mix of legacy applications that are not containerized, along with a bunch of recently added applications that are fully Dockerized. You might have a mix of applications written in Node.js or Go, and using different databases, queuing mechanisms, or deployment endpoints.

The rapidly changing landscape also means that any toolchain decisions you make today will need to be revisited tomorrow, at least for new applications, if not for existing ones. This makes it difficult to commit to a toolchain and over time, you will have many diverse applications with a toolchain "matrix from hell".

You need a way to manage all these heterogeneous applications in a consistent fashion, without needing to rip-and-replace your workflows.

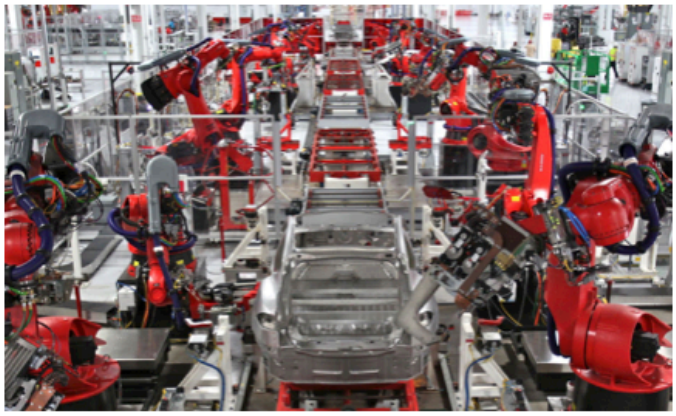
Hyper-efficient DevOps with Assembly Lines

At a macro level, the problems described above are not unique to software development. The driving force behind this rapid evolution is that organizations now realize that software innovation is the only path to long-lasting success, and have less tolerance for bottlenecks to faster innovation.

The car manufacturing industry went through a similar evolution and perfected a highly efficient approach as it went from highly custom cars created manually, to automated parts production with manual assembly, to fully automated assembly lines. It is time to consider a similar approach towards DevOps and software development. We call this approach **"Assembly Lines (AL) for DevOps"**.

manual

partly automated



assembly lines

Assembly Lines (AL) offer a level of sophistication that goes much beyond ARA in managing the complexity of shipping modern applications by connecting “islands of automation” into holistic, interconnected workflows. A simple way to think about it is to imagine an interconnected network, or ecosystem, of pipelines that can exchange data and collaborate with each other as required. State information, such as IP address of provisioned machines, public keys of an encryption service, or the date of last Database backup, is maintained by the Assembly Lines and available to all pipelines that are a part of the network. Collaboration between different DevOps activities and pipelines is through this network, and not through people.

The Assembly Line approach enables automated pipeline-to-pipeline collaboration that extends beyond just an activity, discipline, or team. Any change in one pipeline can be easily propagated to all dependent pipelines without needing human interaction. For example, if your Security Ops (SecOps) team triggers a pipeline to fix a critical security hole in the Operating System, this action can further trigger pipelines that update the entire IT portfolio in a streamlined fashion.

With Assembly Lines, people can collaborate on a much larger scale versus traditional methods of cultural collaboration like meetings, stand-ups, slack channels, email, etc. The larger your Assembly Line network gets, the more value it brings to the teams that are attached to it, which

translates into faster, predictable releases. You will achieve the holy grail of Continuous Deployment when all teams and processes are connected to this network and manual intervention isn't necessary to ship a new version of your application(s).

Benefits of Assembly Lines

Let us take a deeper look at how the Assembly Line approach is different from ARA and other methodologies, and how it addresses the current challenges with DevOps and Continuous Deployment. Using Assembly Lines provides the benefits listed below.

#1: True Continuous Deployment (or Delivery) with interoperability

There are many tools that help you automate individual DevOps activities, but the real challenge is in connecting them and achieving a continuous workflow. Assembly Lines help you achieve Continuous Deployment by connecting your DevOps tools and pipelines into interconnected workflows that retain and exchange information as needed, and minimize the need for human intervention.

#2: Rapid onboarding and scale with "as-code" philosophy

ARA tools today are UI-driven, so end-to-end workflows are unwieldy to configure, manage, or reuse. The workflows themselves aren't versioned, so replaying a previous workflow is not supported.

Assembly Lines use a declarative YML based language that is easy to read, learn and author. They follow the "everything as code" philosophy that is fundamental to modernizing IT. This allows you to quickly start sequencing your DevOps activities into interconnected assembly lines that are versioned and easy to replay. Workflows can be reused easily to define additional workflows, so onboarding becomes very simple.

#3: Powerful nested visibility

As we mentioned earlier, teams today exchange information across pipelines through human interaction like emails, meetings, spreadsheets, or slack conversations. This is inefficient and creates a dependency on people.

With Assembly Lines, every team is aware of activities and information across all pipelines. For example, imagine that a new security patch was applied to the machine image creation pipeline. Even though the SecOps team might take 2 weeks to release the patch, the Test and Ops teams already know it's coming up soon and can better prepare for it. Each team has access to the information they need, such as the latest version of OS running in production,

subnet address of each environment, version of the application running in production with all the changes that are included in that version, how many instances of each application are running, etc.

Information is power, and Assembly Lines provide all teams the ability to have complete visibility into latest and historical information.

#4: Support for bi-modal applications

Every organization has a mix of legacy and modern applications, along with a heterogeneous mix of languages and tools. Assembly Lines enable a consistent workflow for shipping all applications, regardless of their nature. All these applications are a part of the network, enabling information exchange across bi-modal boundaries.

#5: Native Integrations

The toolchain for DevOps is rapidly evolving, and it is often difficult to adopt newer tools because it creates too much overhead to re-configure and manage dependencies.

With Assembly Lines, you get native integrations to all popular tools and languages and can incorporate them into your organization easily. For example, if you want to move to a newer tool, it can be easily attached to the network and reconfiguration is as simple as a few lines of YML. You can also reuse a pipeline built by another team as a template for integrating the tool.

#6: Team-Based Business Intelligence and Analytics

Since Assembly Lines capture information across teams, business intelligence (BI) can be built-in and metrics can be tracked across teams. You can start following a “continuous improvement” model with Assembly Lines providing the benchmark data required to evaluate and improve all connected teams.

Final Thoughts

The main goal of Assembly Lines is achieving true Continuous Deployment by automating interoperability between different DevOps activities. You can use Assembly Lines is to automate the entire process of building and delivering software to your customers, irrespective of toolchain, architecture, or package type.

As automation becomes the default mindset, Assembly Lines provide a new paradigm where programmatic access to information is more critical than the person who can provide it. This ability to exchange information and be notified when dependencies that are two or more degrees of separation away change will be second nature to the new wave of DevOps.

Cutting-edge modern companies like Facebook, Amazon, Google and Netflix have already built homegrown versions of Assembly Lines after spending years of effort and millions of dollars. Like anything else in the world, the best innovation happens in a few companies and is then commoditized and made available to organizations of all sizes and abilities. We believe it is now time for this approach to become the default as we enter the next wave of DevOps.

Appendix

DevOps Terms

- **Continuous Integration (CI)** is a development activity that requires developers to integrate all source code into a shared repository several times a day. Each commit is then verified by an automated build, allowing teams to detect defects early in the cycle. The result of a CI step is typically a unit-tested and consistent codebase.

- A **Pipeline** is a point-to-point set of instructions or activities that are executed in sequence and achieve a singular goal, such as deploying a commit to an endpoint, pushing a Docker image to a hub, or provisioning infrastructure for an environment. For example, a typical **CI Pipeline** can consist of the following DevOps activities - Build, Package, Unit Test and Push a container image with an immutable tag to a hub. An **Infrastructure Provisioning** pipeline can consist of provisioning or updating environments every time the provisioning scripts change.

- **Application Release Automation (ARA)** helps take an application release from source control, through multiple pipelines like build, CI, package, deployment to Dev environment, deployment to Staging environment, and ultimately deployment to Production environment, with a linear workflow. You can configure manual or automated steps or gates between each pipeline.

- **Continuous Delivery** is a philosophy which dictates that the HEAD of your application codebase (e.g. latest commit to master branch), or the most recent immutable version of the software package, is always deployable to production. Continuous Delivery stops just one step before Continuous Deployment, since you ensure that each change is deployable, even if you don't choose to deploy it. To achieve Continuous Delivery or Deployment, you need to automate every activity in your DevOps workflow.

- **Continuous Deployment (CD)** is achieved when you can do zero-touch deployments from source control to production for every commit. This should be the goal for every application and offers the fastest way to release software updates to your customers.

shippable

Shippable is a DevOps platform that helps you ship code faster and more reliably by unifying all DevOps tools and activities into **Assembly Lines** with complete visibility, traceability, and auditability.

More information

1. Shippable home page: www.shippable.com
2. Documentation: docs.shippable.com
3. Pricing: www.shippable.com/pricing.html
4. Shippable Server: www.shippable.com/enterprise.html

Contact

1. Contact form: www.shippable.com/contact.html
2. Sales: sales@shippable.com
3. Support: support@shippable.com

Social

1. Twitter: @beShippable
2. LinkedIn: https://www.linkedin.com/company/shippable
3. Facebook: https://www.facebook.com/beShippable