# Lecture 12 – DevOps Assembly Lines

We’ve opened up the toolbox of a DevOps engineer this course and have looked all commonly used components of the DevOps toolchain. We’ve seen:

* How developers can leverage CI for rapid development
* Ops provisions and configures infrastructure
* QA uses test automation suites
* Automation of security patching
* Semantic version and release approval for product managers
* Deploys for multiple environments

When we focus on all of the above, we can connect each activity which are typically performed by independent teams. DevOps assembly lines focus on just that, the ability to glue together each task to achieve continuous delivery.

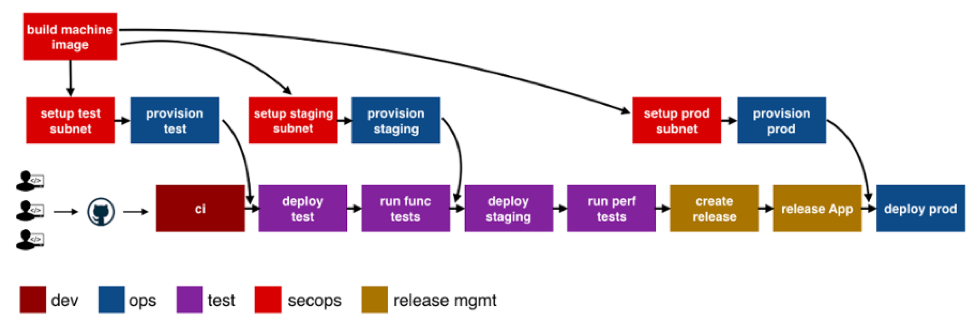
It’s quite common in the field to see most teams adopt one of the following approaches:

* Using project management software like Jira. For example, when a new version of an application is ready for release, the test team will typically assign a Jira ticket to a member in Ops. The ticket is intended to let them know feature requests have been qualified and a production deployment is needed for the changes to go live. Ops would then copy the release information into a deploy script and run it.
* A DevOps engineer is responsible for writing a Jenkins job which handles automated tests and then pushes application version information to an S3 bucket. Another Jenkins job polls the S3 bucket and when a change is detected, the deployment scripts are executed.

It should be apparent which of the above two is more efficient. One requires a lot of manual steps performed by humans which requires communication and sometimes more communication than necessary. Often, back-and-forth email exchanges or instant notifications will be needed to glue the process together. The second approach works better but only works well for small teams working on a single application. It can produce spaghetti-code like automation scripts when you have many applications or a single application using microservices.

Enter DevOps Assembly Lines (AL). AL addresses this problem by focusing on gluing together these various activities into one streamlined, event driven workflow. This provides the ability to easily share state and other information across different activities.

The ability to glues these activities together into their own silo is best represented with the following image:

Take note that each box is its own pipeline and the continuous integration process is just one activity in the assembly line. Also note that each box has its own specific needs with respect to tools, runtime, configuration, notifications etc. An advantage to using an assembly line to formulate all the activities is that each team can take ownership of a specific pipeline in the assembly line. Lets take for example a change by Ops in a provisioning script for a staging environment. For this to take place, the test team needs to redeploy the application and run performance tests.

To provide an accurate and succinct definition for an assembly line, an assembly line is a pipeline of pipelines. A DevOps assembly line needs to support the following:

* Ability to easily define workflows across multiple pipelines
* The workflows must be versioned and reusable to enable rapid changes and scaling for multiple applications and/or microservices
* Integrations with all popular VCS, clouds, artifacts, DevOps tools, languages and services
* Ability to pass state and other information while triggering dependent pipelines
* Automatic triggers or manual approval gates between each pipeline
* Configurable notifications for each stage of every pipeline
* Release automation features such as semantic versioning of packages
* The ability to create an audit trail for each pipeline with the ability to go back or forward to a specific state
* Abstraction of all sensitive information like tokens, passwords, keys etc.
* Roles and permissions restricting assembly line and pipeline actions
* Manage DevOps infrastructure, including spinning VMs and containers up and down as required
* Visibility into each pipeline and stage, including logs, status and versioned data
* Intelligently collect metrics and analytics across pipelines to help identify bottlenecks

To summarize, DevOps AL helps automate and scale end-to-end workflows across all teams regardless of the tooling required.