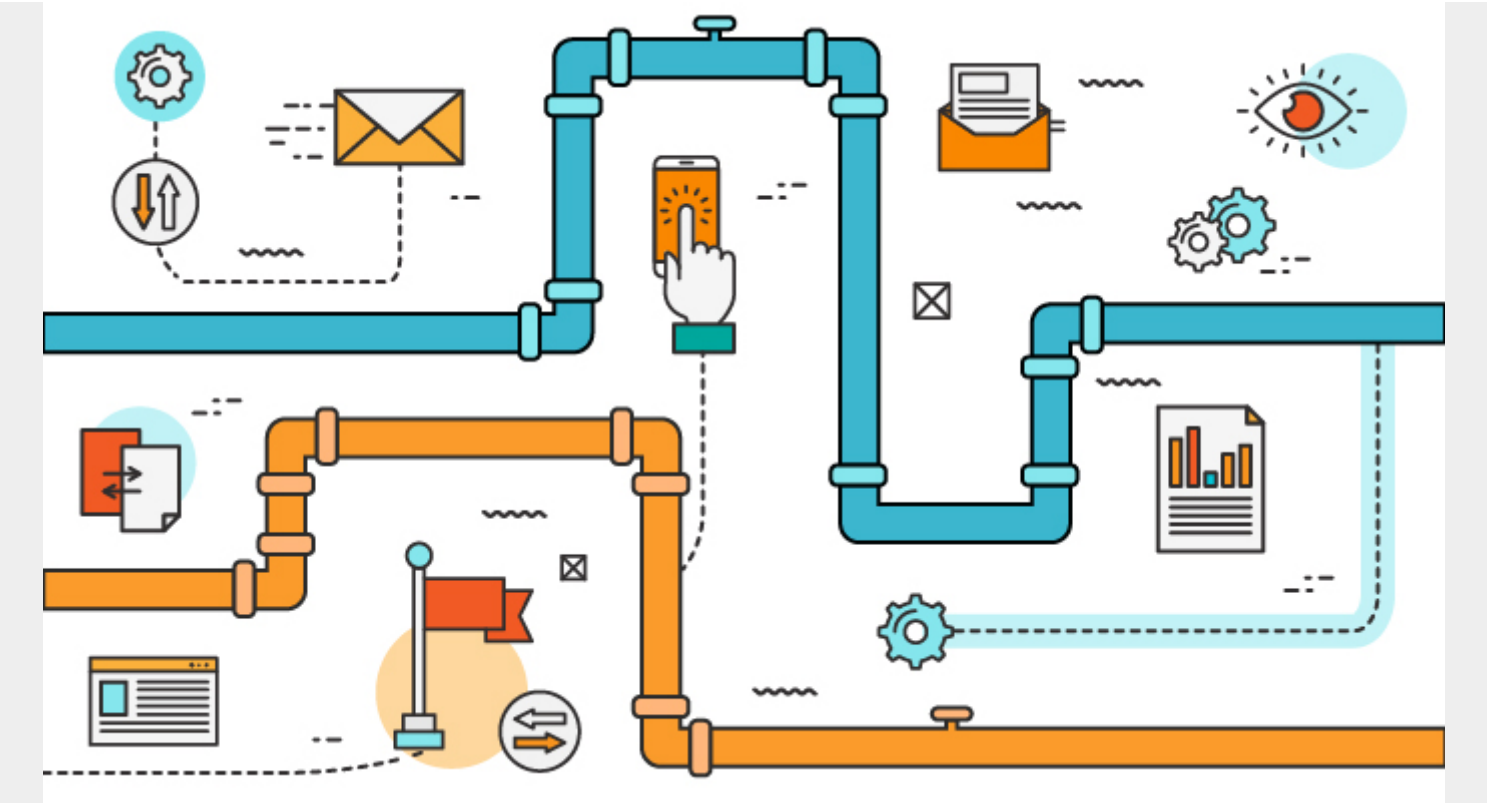
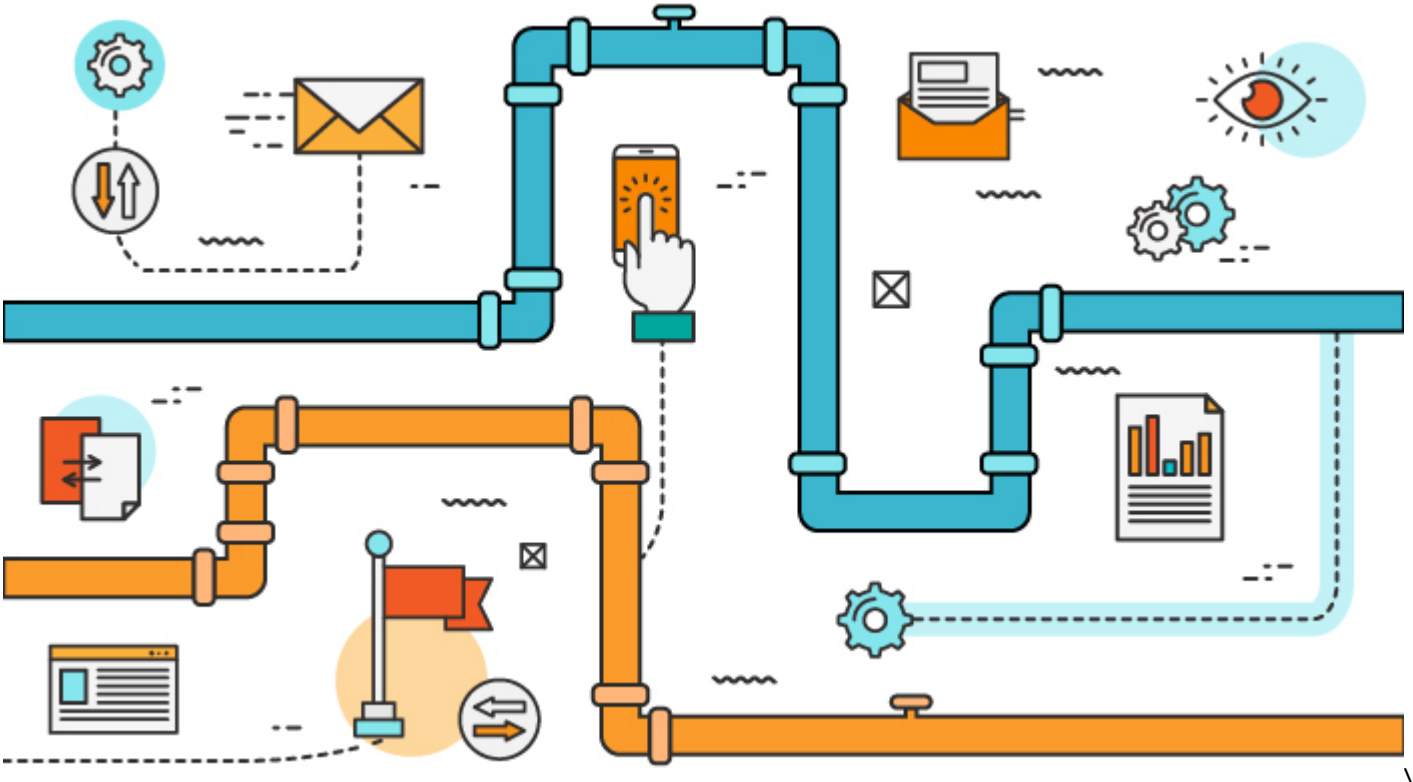
HOW TO BUILD A CD PIPELINE



This is the second blog in our mini-series that illustrates how BMC was able to use the Spinnaker continuous deployment platform to deliver a new cloud-native product that we push to production once a week.

Be sure to also read: Getting Started with Cloud Native Applications, Infrastructure and How "Everything as Code" changes everything, and Five Best Practices for Building Security as Code into a Continuous Delivery Pipeline.



We had a big dream for the cloud-native SaaS application that we were building – to deploy it like Google, Facebook, and Netflix – by pushing hundreds of production changes each week. We started with -our vision of the product as well, our requirements for a high-velocity continuous deployment pipeline, and awareness of the continuous integration/continuous deployment (CI/CD) tools in marketplace.

We decided to start with the Spinnaker Open Source Software tool and within a short three week period, a two--person team had built eight pipelines using the Spinnaker CD tool, one for each of our 8 microservices, and we began actively pushing software to the Amazon AWS cloud on a weekly basis. This blog describes our DevOps CI/CD journey and key learnings and best practices for achieving continuous delivery with Spinnaker.

Our requirements for a continuous deployment pipeline

Our overwhelming objective for CD was to create a repeatable, safe way to deliver software to production as quickly as possible while ensuring confidence and stability. We wanted to automate our pipelines, integrate with a wide array of test automation tools, and create dynamic environments on AWS cloud during various stages of deployment. We also wanted visibility into the status of all our pipelines and environments in development, QA and production. Finally, we wanted near zero-downtime capabilities in production, which could be accomplished using techniques such as canary, rolling, and Blue-Green deployment and rollback strategies.

Making the decision – Spinnaker vs. Jenkins

We had several decisions to make. We picked GitHub enterprise for managing our code and used Jenkins for building software and CI,

For our Continuous Delivery (CD) pipeline, we debated between using Spinnaker and Jenkins. Jenkins can be used for the CD pipeline; however, managing hundreds of job chains becomes quite

complex with Jenkins and it also does not have advanced deployment and cloud capabilities.

After investigating Spinnaker, we found that it easily scales, supports application-centric advanced deployment strategies out of the box, has good pipeline visualizations, and supports management of multiple dynamic cloud environments. Based on our experience using Spinnaker over the past six months, we feel confident that it has significantly helped us to achieve our original goals for continuous deployment.

Best Practices for CD

Here's an overview of best practices for CD based on our experiences using Spinnaker and running dozens of microservice pipelines for our cloud-native application.

1. **Plan for frequent updates for each microservice pipeline**

An application should be designed to contain many small microservices. Each microservice should have its own deployment pipeline so that it can be *independently* and *frequently* updated in production. Typical stages in a deployment pipeline are: build, unit testing, integration testing, which includes functional, API, performance and security tests. Each of these stages can also include creation of dynamic environments in the cloud so that it the stage can be provisioned, executed and decommissioned as a part of the pipeline. Spinnaker has a number of built-in cloud plug-ins, so creating and destroying environments in AWS, Google and other clouds can be done easily.

2. **Declaratively specify application microservices as Infrastructure as Code**

Each microservice is specified in terms of its application and infrastructure stack. This can include the application Jar file, Docker file or server image such as an AMI, any related service such as AWS Lambda functions, plus all configurations, policies, etc., all managed in an infrastructure-as-code manifest. This process ensures versioning, consistency, change auditing, and testing. Also, the build artifact resources (such as baked images, JARs, AWS Lambda functions) produced are consistently passed among all of the stages of a pipeline from test to production, eliminating the risks of varying environments or configurations. We used AWS CloudFormation templates as our infrastructure-as-code manifest and tested these through Spinnaker pipeline stages. Our third blog will describe this practice in more detail.

3. **Visualize pipelines and environments**

Live visualization of the pipelines, environments and the versions or build numbers of microservices running in these environments allow both developers and operations teams to have a shared common view and fix issues as soon as something "breaks" in the pipeline, such as failed QA tests or failed security tests.

4. **Early left-shift security**

Security should be done as early as possible in a lifecycle and must be automated as a part of the DevOps CI/CD process. Using Spinnaker stages, we do penetration testing and security testing of our templates and environments before the code gets into production. Learn more about this in our fourth blog on how we incorporated security and compliance as a part of our pipeline.

5. **Test automation with prioritization and early feedback**

Automating tests is perhaps one of the most critical aspects of pipeline design. Without full automation, high quality and coverage of tests, the deployment pipeline will result in production failures. The "doneness" criteria for each sprint requires comprehensive test automation in a production-like environment. Of course, in reality time is limited so you should prioritize automation around the critical capabilities and flows, and then reevaluate and prioritize cost-benefit of additional automation.

6. **Use staging before production**

The data and environment for testing and automating should mirror production as close as possible. Most pipelines have a "pre-production" stage before pushing software into production. We built multiple stages across many AWS regions with Spinnaker before pushing the software to production.

7. **Advanced Deployment Strategies**

One of our key goals is to update software in production very frequently and without any downtime. This is an area where Spinnaker excels by easily managing multi-region deployments, such as Blue-Green and canary deployments for server groups and clusters across AWS and other public IaaS clouds. Our cloud-native application was based on AWS PaaS services, like AWS Lambda and AWS Beanstalk, and Spinnaker leverages these services to provide similar functionality.

8. **Monitoring user experience and metrics**

Be sure to continuously get feedback about the user experience, response times, performance monitoring of key business services, and technical metrics for all environments from development, QA and production. Keeping an eye on these metrics and making them part of your pipeline stages will ensure that no degradation happens as you push out releases with increasing frequency.

9. **Culture – Developers own it end-to-end**

Culture plays a very important part of our process of agile development. Just having the right tools and technology doesn't cut it. We indoctrinated a new set of developer responsibilities. This includes owning not just code but also owning the automation, pipeline and production operations for each microservice. This mind shift is critical to successfully adopting an agile, continuous delivery and deployment process.

While Spinnaker OSS is off to an amazing start, BMC discovered several key gaps in operationalizing Spinnaker that we are currently addressing within our development team. If anyone is interested in talking about what we are what we are doing to enhance Spinnaker, please contact spinnaker@bmc.com

Stay tuned for our next blog on about how treating Infrastructure as Code helped drive quality and consistency in our development pipeline.