Lessons Learned:
Deploying Microservices Software Product in Customer Environments

Mark Galpin, Solution Architect, JFrog, Inc.
Who’s speaking?

Mark Galpin
Solution Architect
@jfrog

LinkedIn
Twitter
magalpin
Microservices are Cool
But what about installation
A word on the difference between SaaS and On Prem

OWNERSHIP
A tale of three products

A legacy (10 years now!) monolithic web application architecture written in Java deployed on premises at thousands of customers world-wide

A java-based microservices architecture application designed for the cloud, operating for 5 years, transitioning in 2018 to be offered on premises. Hosts jcenter and homebrew, easily scales to billions of downloads a month

A greenfield project in 2016/2017, designed as an on-premises microservices architecture from the start.
So what are we going to talk about?

• Discussion of the artifactory experience, what did we learn migrating a legacy monolith to cloud-native?
  – Major version upgrade was required to modify architecture
• Discussion of the xray experience. Microservices design on prem: How did it go?
• Based on this what are we taking forward into the next one?
A place to start: Artifactory HA
A first step: VM orchestration

• Lesson: Separation between the application layer, and the configuration layer.

• Example: health check (/api/system/ping),
  – What is it: confirms artifactory web service, and connection to DB and filestore. Best way to test
  – Problem: It required an authenticated user
  – Solution: Make an anonymous version of this available
VM orchestration

• The importance of startup scripts
• Issue: Health check starts before application starts
• Solutions?
  – Complex script to try to prevent health check from initiating until successful start
  – Just wait long enough
A first step: Early containerization

• Install the artifactory RPM in a container and we’re done, right?

• What about HA?
  – Different directory structure!
  – Take default image and “customize”
  – NFS requirement?
The first Mesos implementation

• First container based self-healing orchestration of Artifactory
  – Thank you mesos team for much assistance!
  – Able to leverage existing mesos capabilities for most things
  – Able to leverage mesos-based DB services

• Issues:
  – User MUST supply an external NFS mount
  – License management required an extensive hack
Enter Artifactory 5!

- Artifactory 5 objective was to take lessons learned from previous cloud-native work
- First cloud-native ready version of artifactory.
- Major changes:
  - Config for HA no longer requires shared storage
  - Creates mechanism for node cross-talk to share config and cached artifacts
  - No more NFS!
  - License management for clusters shifted to the application layer
Where we are today?

- Cloud-native deployments of artifactory for Mesos, Kubernetes, Docker Swarm
- Broke the first microservice out of the monolith into a second web-application service
So what about xray? Back in time.
Lesson 1: Don’t overestimate the customer

• We released Xray with a cool docker-in-docker install script and as a set of docker containers.
• First request from customers: “Do you have an RPM install of that?”
Lesson 2:
Start like you mean to go on

• System was architected for enterprise/HA/etc.
• For the 1.0 release, we built/test it only with the default architecture of all containers created on one server
• It took most of a year to find all the issues this caused for us to enable HA and horizontal scalability
• DevOps!
Lesson 3: Flexibility!

- Nearly all requests made have been to give customers more flexibility on install
  - Ability to specify custom paths
  - Bring-your-own infrastructure
  - Source of docker containers
Lesson 4: What about startup scripts?

• Startup order:
  – Microservices are stateless
  – But they do have dependencies in order to function
  – Xray startup script explicitly checks dependencies and introduces a startup order
  – Makes it easier for a customer less familiar with microservices architectures to understand startup process.
Moving forward: Bintray on Premises & JFrog Platform
What does platform look like?
What are we doing with this one?

• Simple is better
  – Consolidate infrastructure services across products
  – Use small services for scalability & flexibility, but try to keep the total number down for on-premises

• Start with the enterprise architecture deployment
  – If you don’t honor scalability/flexibility at the beginning its harder later

• Start with a container-orchestration implementation. Understand we can’t end there.
Most Important Takeaway

• DevOps!
  – Developers and Packaging teams need to be working on deployment/packaging problems from the beginning!
Thank You!

• Q&A

• By the way, we’re hiring!