Beyond CI to Production Scale PaaS with Docker

dockercon 15

SF JUNE 22-23

Platform Engineering @ PayPal

- 165 Million active PayPal customer accounts
- Presence in 203 markets and 100 currencies
- $235 Billion payment volume
- 12.5 million payment transactions every day

Support ever increasing scale of operations

- Several thousand OpenStack servers across multiple data centers
- More than 3000 PayPal developers supported
- Thousands of application deployments performed every day

Boost developer productivity

Last year at DockerCon 2014…

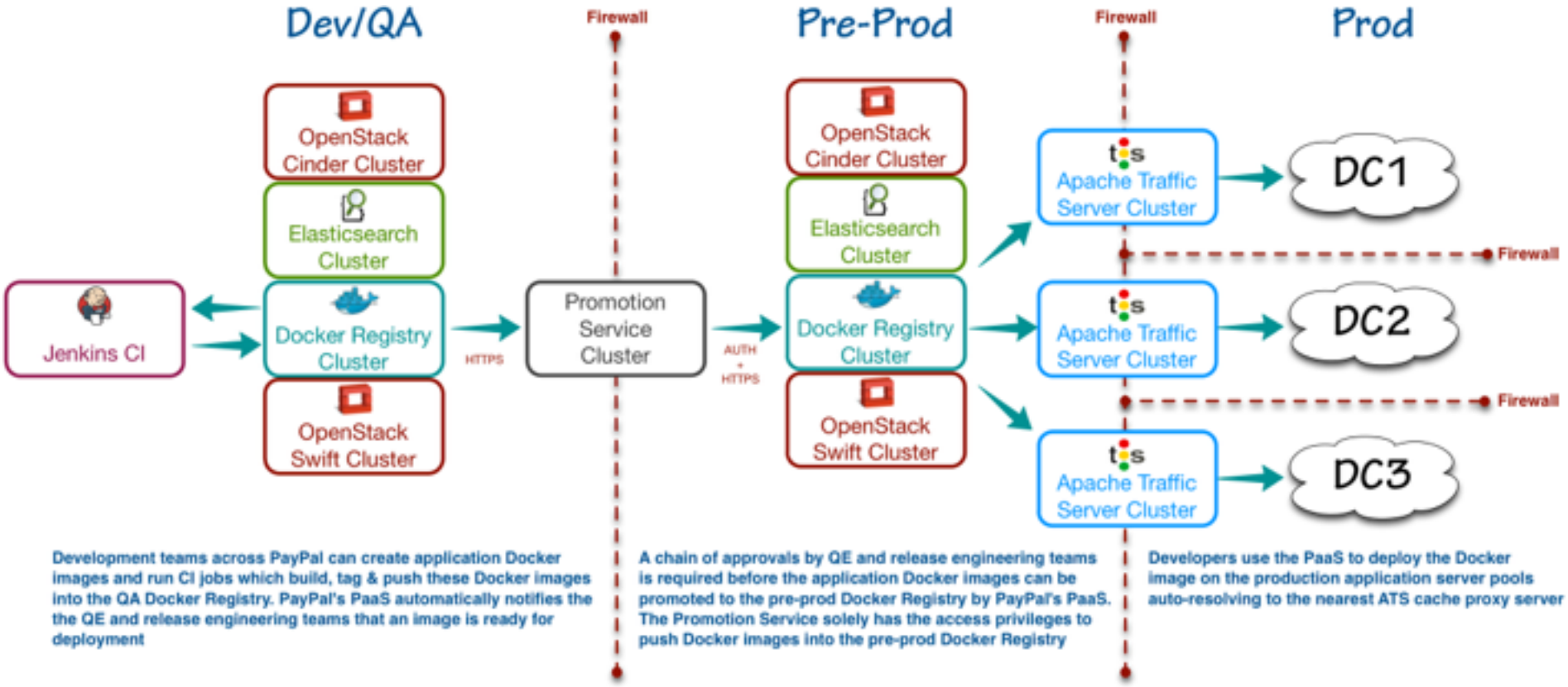
- Thousands of VMs dedicated to run Jenkins!
- Utilization is less than 5% ☹
- VMs idle most of time (no one runs CI builds frequently)

VM sprawl and poor resource utilization

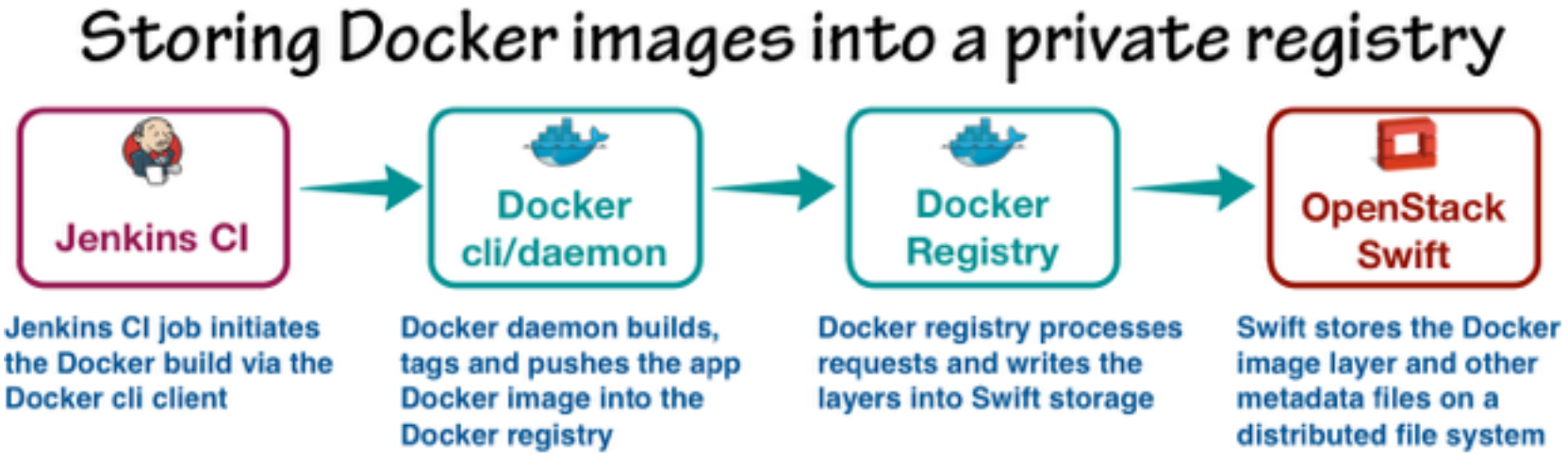
- Polyglot application stacks (Java, C++, Node.js, Python, Scala)
- Different OS flavors (Ubuntu & RHEL)
- Software version conflicts
- Special hardware requirements

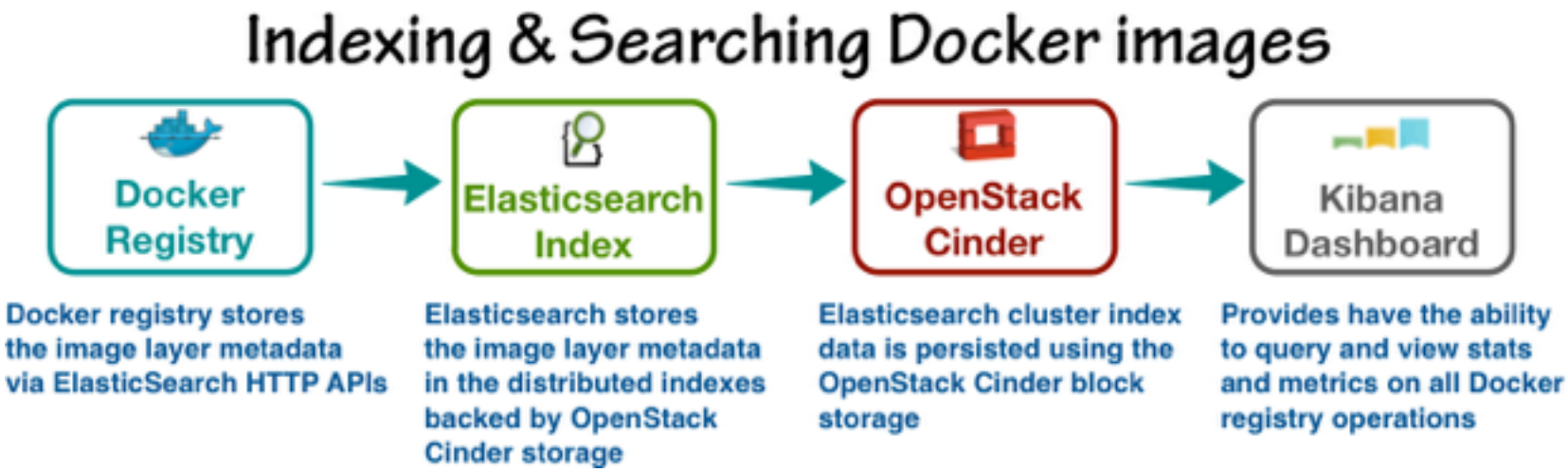
Build slave management and maintenance is a nightmare

How PaaS Orchestrates The Docker PDLC

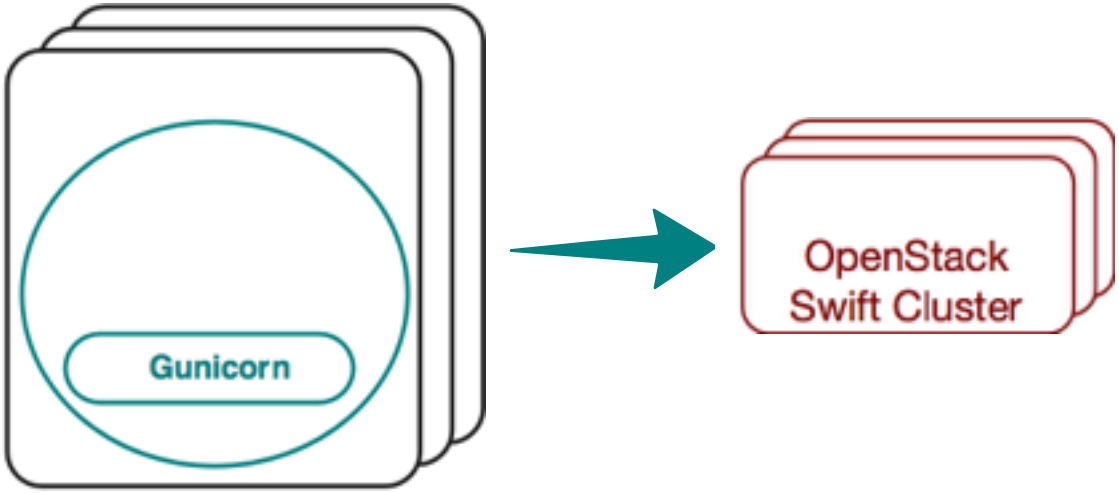


Building & Storing Docker Images





Docker Registry HA Setup



Features

- Supervisord as the process manager
- Logrotate for registry and nginx logs
- Elasticsearch plugin for indexing
- Swift plugin for storage
- Basic authentication
- Ansible playbook for setting up the registry
- HA running behind F5 load balancer
- Docker load used to deploy the registry for the first time
- Swift auto-sync between data centers

Docker Image Index

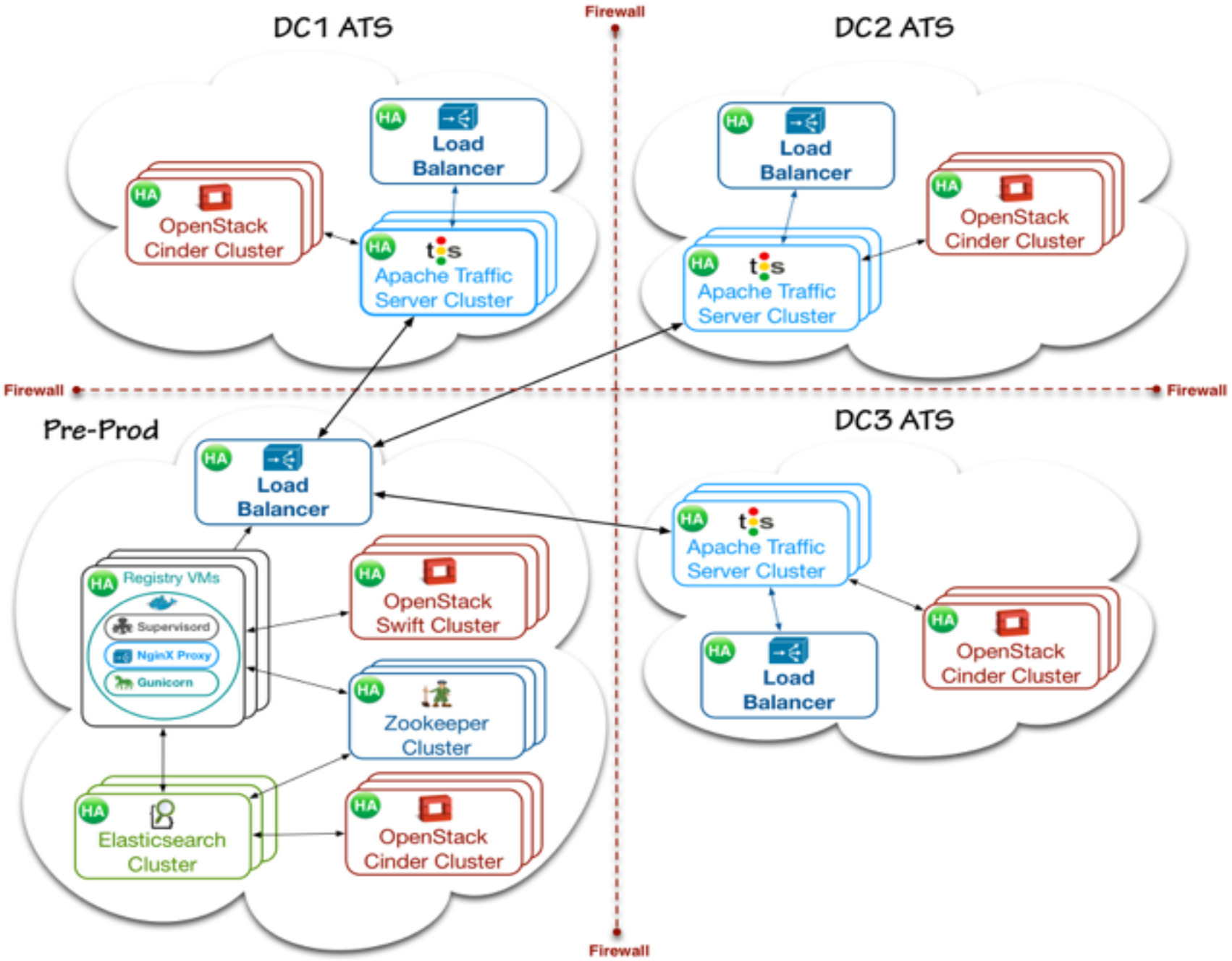
Challenges

- Production firewalls block multicast clustering protocol
- ES sniffing timeout issues when ES nodes were unavailable
- ES split-brain problems with clustering

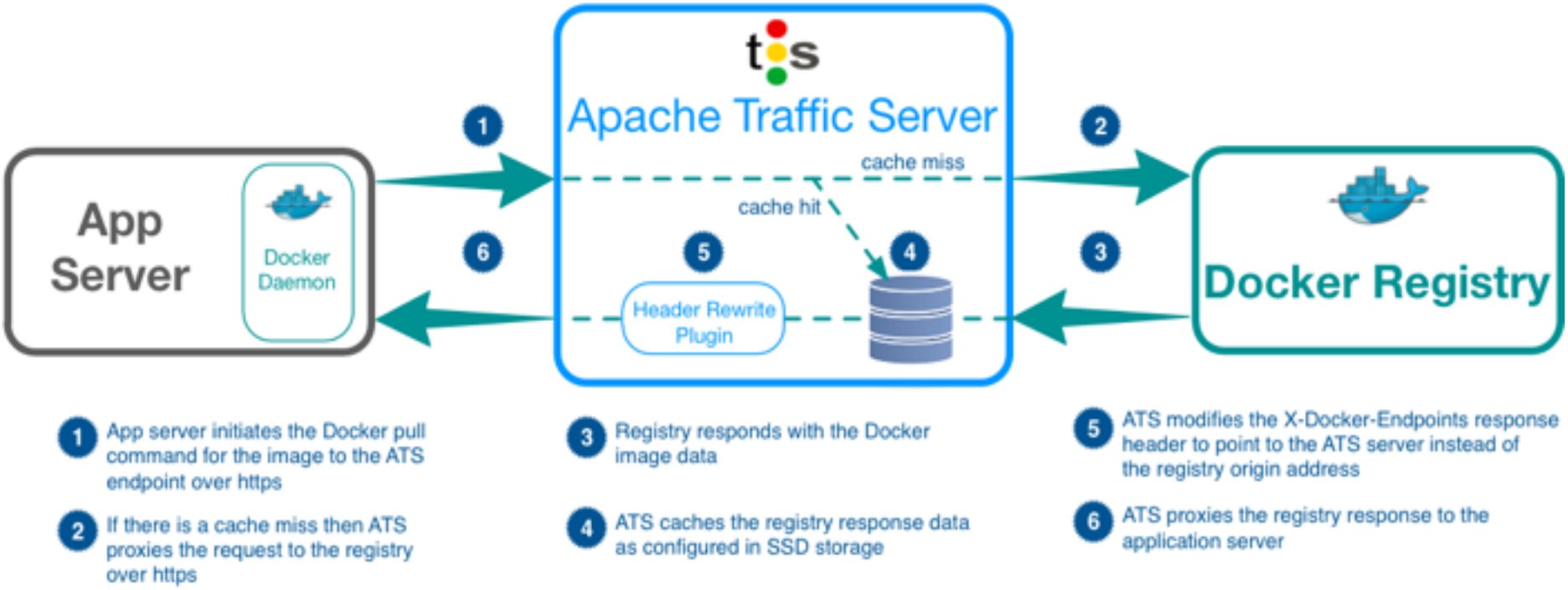
Solutions

- ES indexing plugin for the Docker Registry
- Zookeeper transport plugin for ES Python client
- Persisting ES index data using OpenStack Cinder

Cross-datacenter View



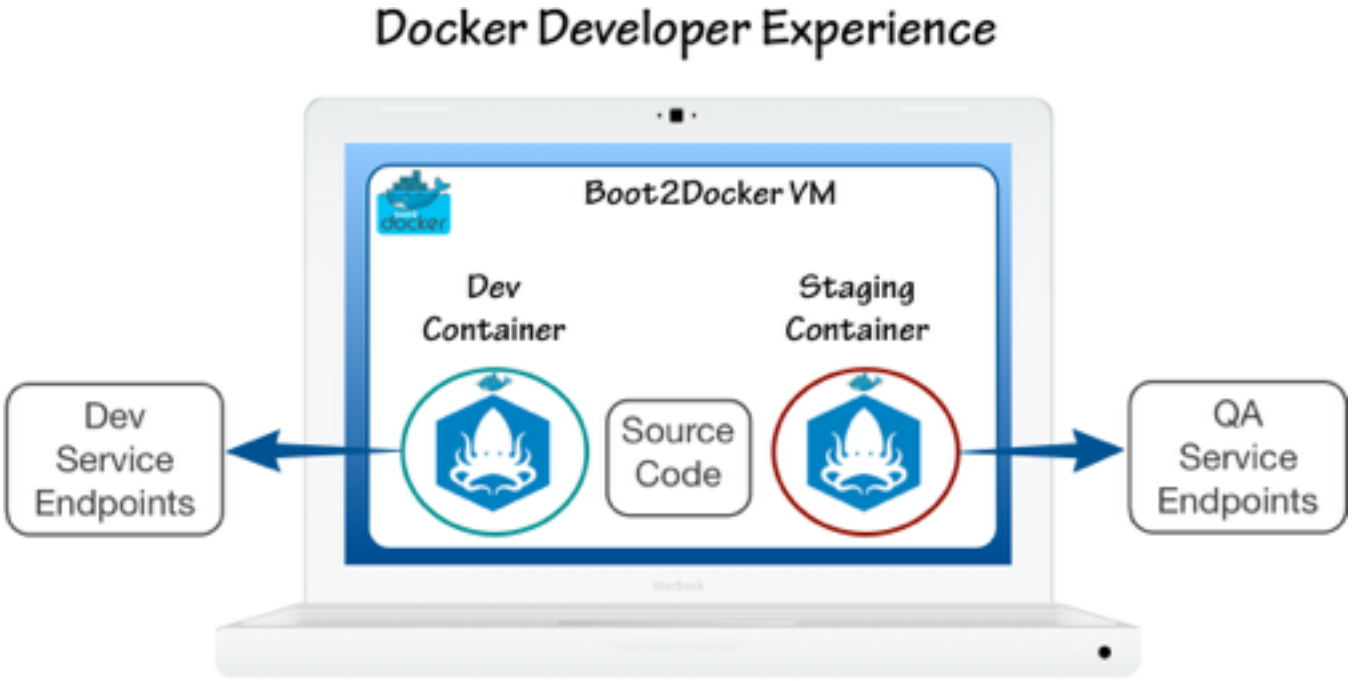
Deploying Docker Images In Production



Features

- DNS-based ATS discovery per DC
- Header rewrite plugin
- Custom Cache rules
- Custom SSL certs
- ATS Ansible deployment

Dockerized Development Environments



- Building an application stack should be simple, but it's not!
- Development environments are snow-flakes
- Development environments should be self-contained

Container with Dev configuration

```
devweb:
  image: private-registry/stacks/kraken_dev
  command: /docker/init.sh
  volumes:
    - ./:/src
    - /src/node_modules
    - /src/.npm
    - /src/.nvm
    - /src/.node-gyp
    - /src/tmpnpm
  environment:
      - NODE_ENV=development
      - DEPLOY_ENV=development
  ports:
    - "8000:8000"

```

Container with Stage configuration

```
stageweb:
  image: private-registry/stacks/kraken_dev
  command: /docker/init.sh
  ports:
    - "80:80"
    - "443:443"
    - "8000:8000"
  volumes:
    - .:/src
    - /src/node_modules
    - /src/.npm
    - /src/.nvm
    - /src/.node-gyp
    - /src/tmpnpm
    - /src/.build
    - /src/.builds
    - /src/target
    - /src/.packageignore_tmp
    - /src/deploylogs
  environment:
    - NPM_CACHE=/src/.npm
    - NODE_ENV=staging
    - DEPLOY_ENV=STAGE
    - BASE_DIR=/src
    - NPM_TMP=/src/tmpnpm
    - NVM_DIR=/src/.nvm
    - NPM_REGISTRY=http://internal.npm.reg
  hostname: boot2docker
  domainname: slc01.dev.ebayc3.com
```

Demo

Thank you

Mohit Soni

Software Engineer

@mosoni

Ashish Hunnargikar

Software Engineer

@hunnarg

#dockercon