Beyond CI to Production Scale PaaS with Docker

PayPal

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

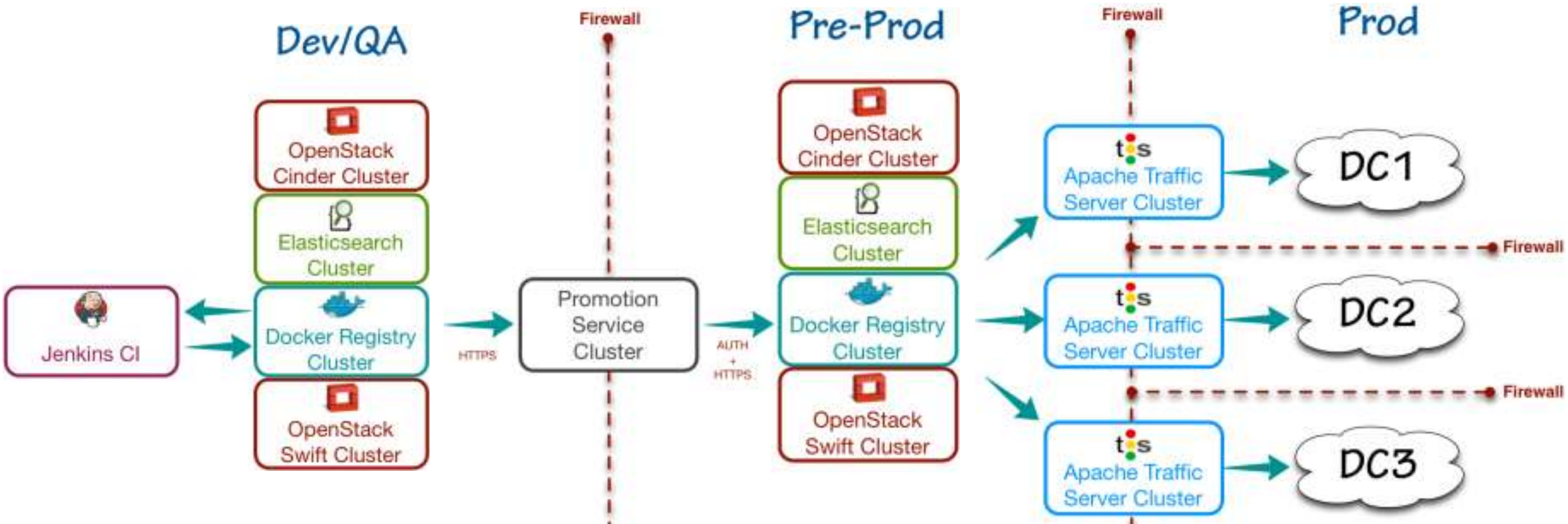
- Thousands of OpenStack instances across multiple data centers
- More than 3000 PayPal developers supported
- Thousands of deployments performed every day

Boost developer productivity

Why PaaS & Docker?

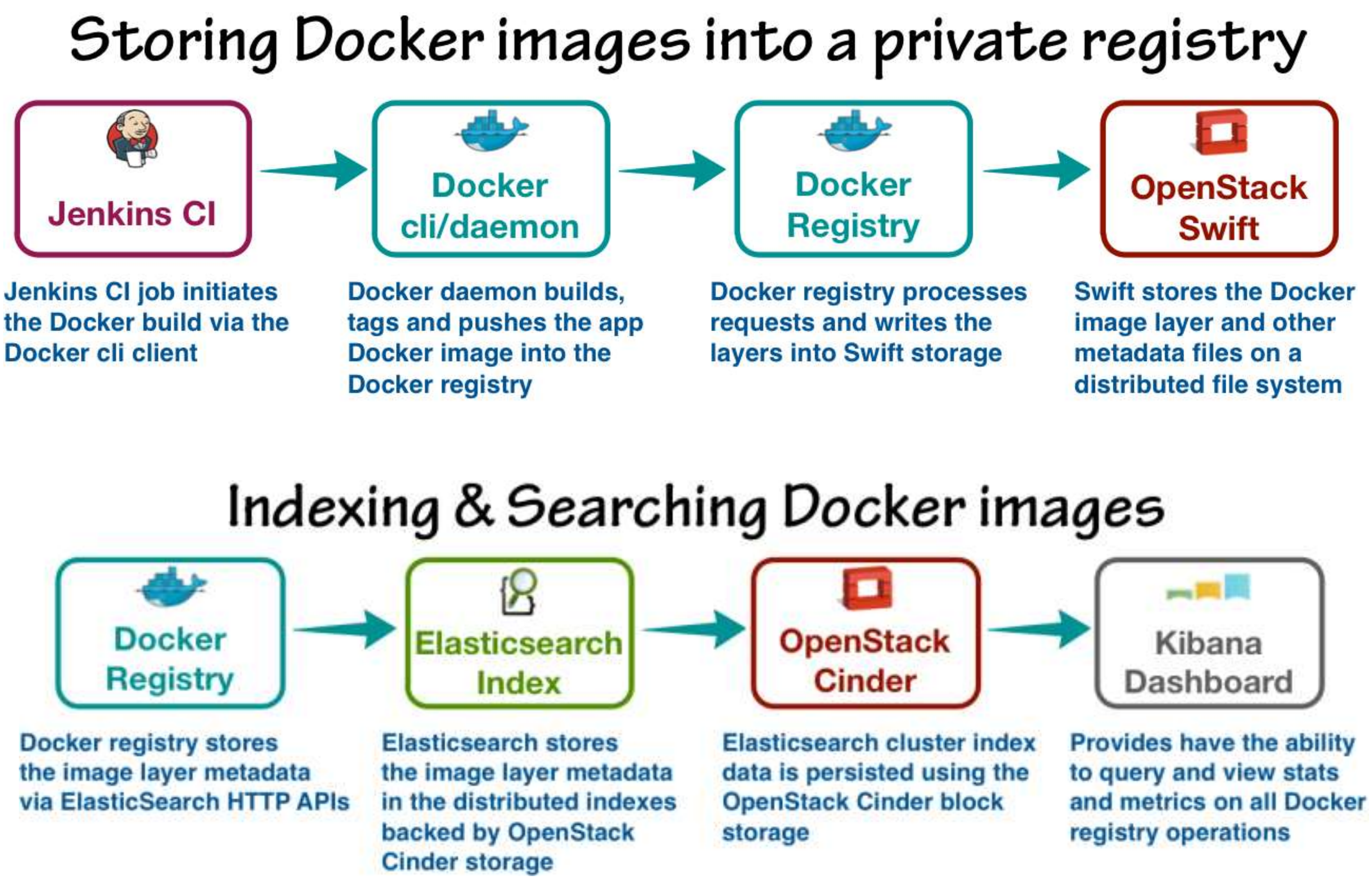
- Runtime workload portability across machines & Infrastructure
- Image based deployment for repeatable outcome
- Support for different OS flavors (Ubuntu & RHEL)
- Polyglot application stacks (Java, C++, Node.js, Python, Scala)
- Eliminates application dependency drifts across machines
- Git-like capabilities for tracking successive versions of a container & history on how a container was assembled
- Rapid application deployment and flex up/down
- Key to incremental CI environment upgrades
- No difference between provisioning & deployment

How PaaS Orchestrates The Docker PDLC



1. Developers create Docker applications
2. CI jobs build, tag & push the Docker images
3. Docker image registered with PayPal's PaaS system
4. QE & Release Engg. approvals required
5. Promotion Service executes docker push
6. Developers deploy into production
7. Docker image downloads auto-resolve to nearest ATS server

Building & Storing Docker Images



Storing Docker images into a private registry

Jenkins CI → Docker cli/daemon → Docker Registry → OpenStack Swift

Jenkins CI: Jenkins CI job initiates the Docker build via the Docker cli client

Docker cli/daemon: Docker daemon builds, tags and pushes the app Docker image into the Docker registry

Docker Registry: Docker registry processes requests and writes the layers into Swift storage

OpenStack Swift: Swift stores the Docker image layer and other metadata files on a distributed file system

Indexing & Searching Docker images

Docker Registry → Elasticsearch Index → OpenStack Cinder → Kibana Dashboard

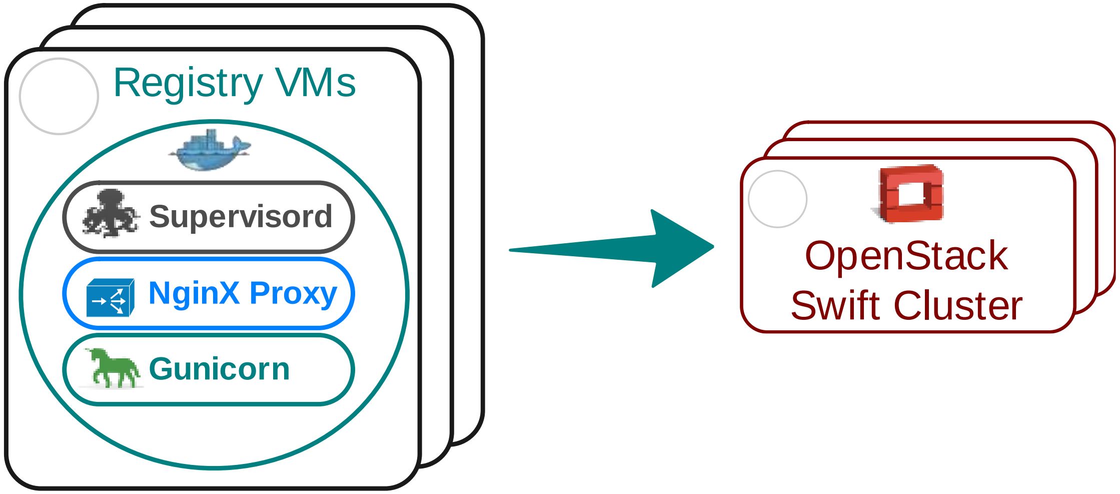
Docker Registry: Docker registry stores the image layer metadata via ElasticSearch HTTP APIs

Elasticsearch Index: Elasticsearch stores the image layer metadata in the distributed indexes backed by OpenStack Cinder storage

OpenStack Cinder: Elasticsearch cluster index data is persisted using the OpenStack Cinder block storage

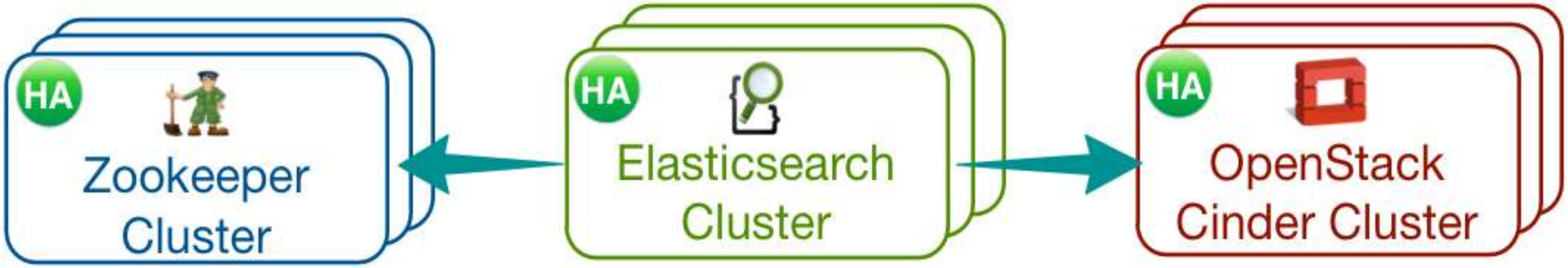
Kibana Dashboard: Provides have the ability to query and view stats and metrics on all Docker registry operations

Docker Registry HA Setup



- 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

Docker Image Index



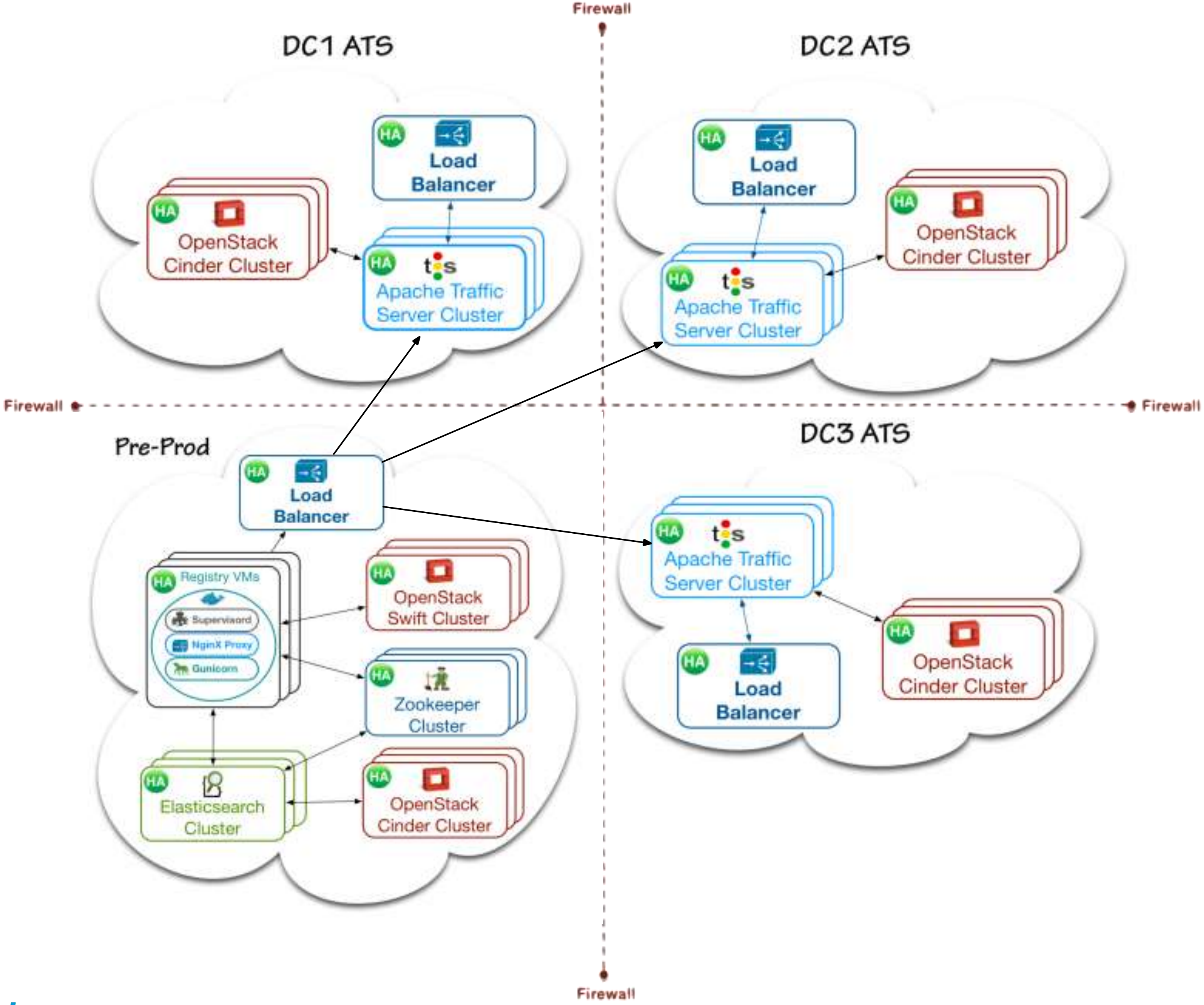
Challenges

- Production firewalls block multicast clustering protocol
- ES sniffing timeout issues and split-brain problems when ES nodes were unavailable
- Docker index/hub is not open source

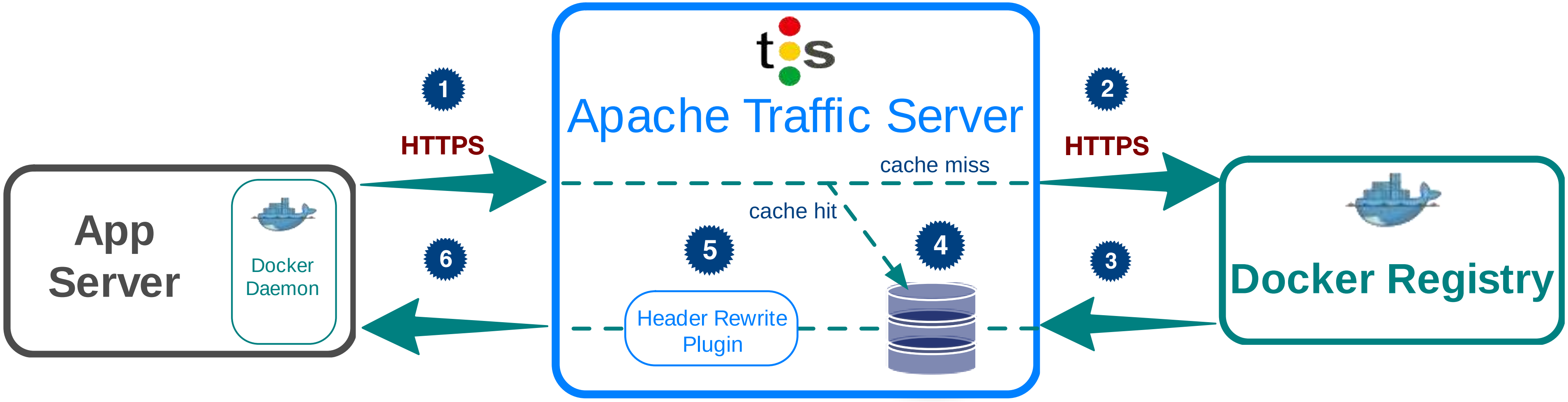
Solutions

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

Cross-datacenter View


Firewall
DC1 ATS
HA Load Balancer
HA OpenStack Cinder Cluster
HA Apache Traffic Server Cluster
DC2 ATS
HA Load Balancer
HA OpenStack Cinder Cluster
HA Apache Traffic Server Cluster
Firewall
Firewall
Pre-Prod
HA Load Balancer
HA Registry VMs
Supervisord
NginX Proxy
Gunicorn
HA OpenStack Swift Cluster
HA Zookeeper Cluster
HA Elasticsearch Cluster
HA OpenStack Cinder Cluster
DC3 ATS
HA Apache Traffic Server Cluster
HA Load Balancer
HA OpenStack Cinder Cluster
Firewall

Deploying Docker Images In Production



1 App server initiates the Docker pull command for the image to the ATS endpoint over https

2 If there is a cache miss then ATS proxies the request to the registry over https

3 Registry responds with the Docker image data

4 ATS caches the registry response data as configured in SSD storage

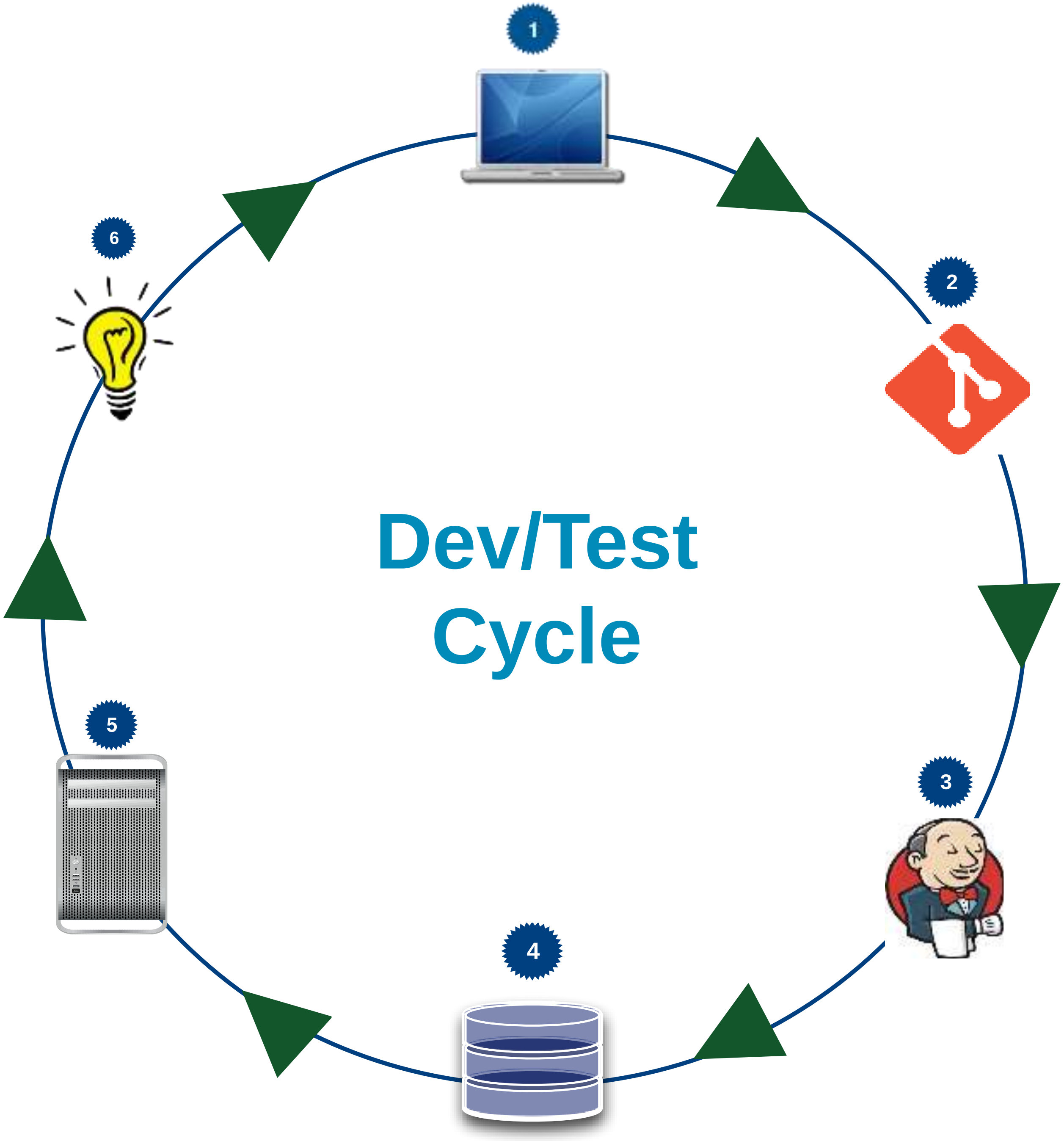
5 ATS modifies the X-Docker-Endpoints response header to point to the ATS server instead of the registry origin address

6 ATS proxies the registry response to the application server

- Custom SSL certs at both ends
- Always serve image layers from cache
- Serves metadata from cache only if connectivity to registry lost
- DNS-based ATS discovery per DC
- Header rewrite plugin
- ATS Ansible deployment

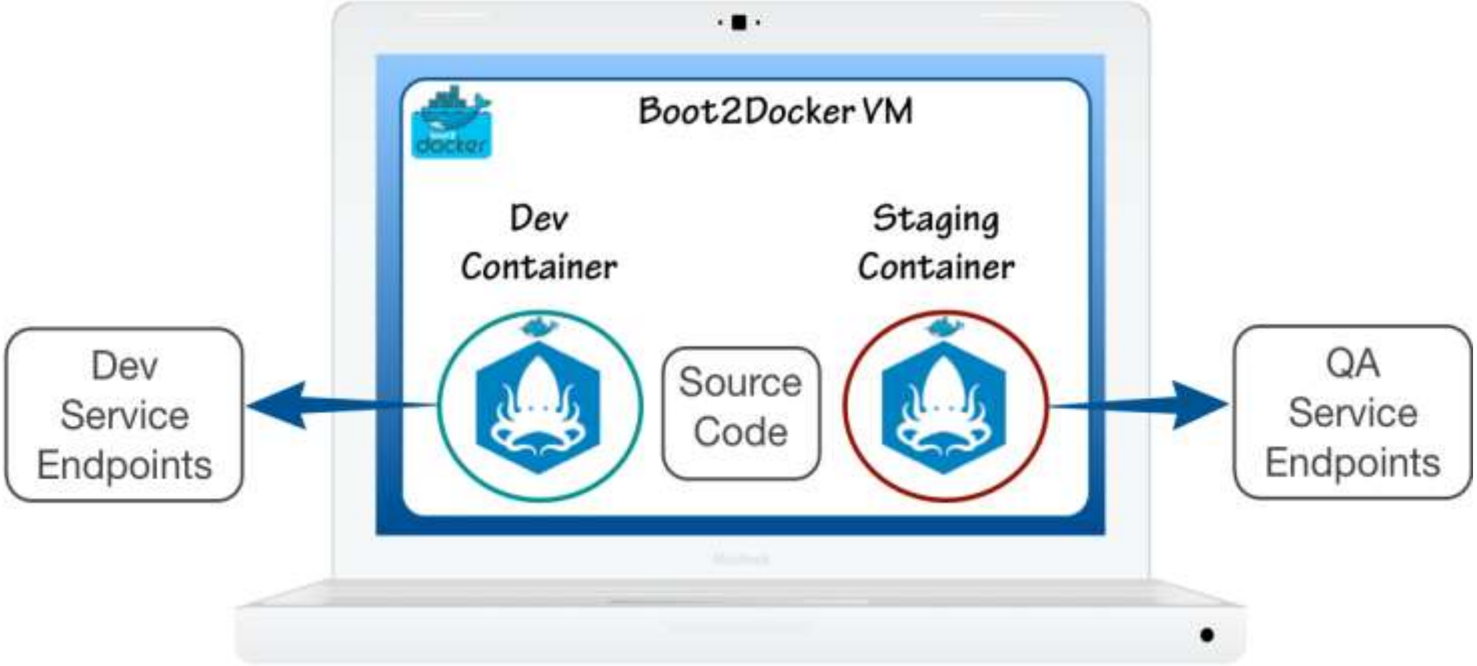
Development Environments

- Building an application stack should be simple, but it’s not!
- Development environments can become snow-flakes
- Development environments should be self-contained
- Difficult to simulate QA/Prod environments from developer laptop



Dev/Test Cycle

Dockerized Development Environments



- ➢ docker-compose
- ➢ boot2docker VM (for Mac and Windows)

Demo

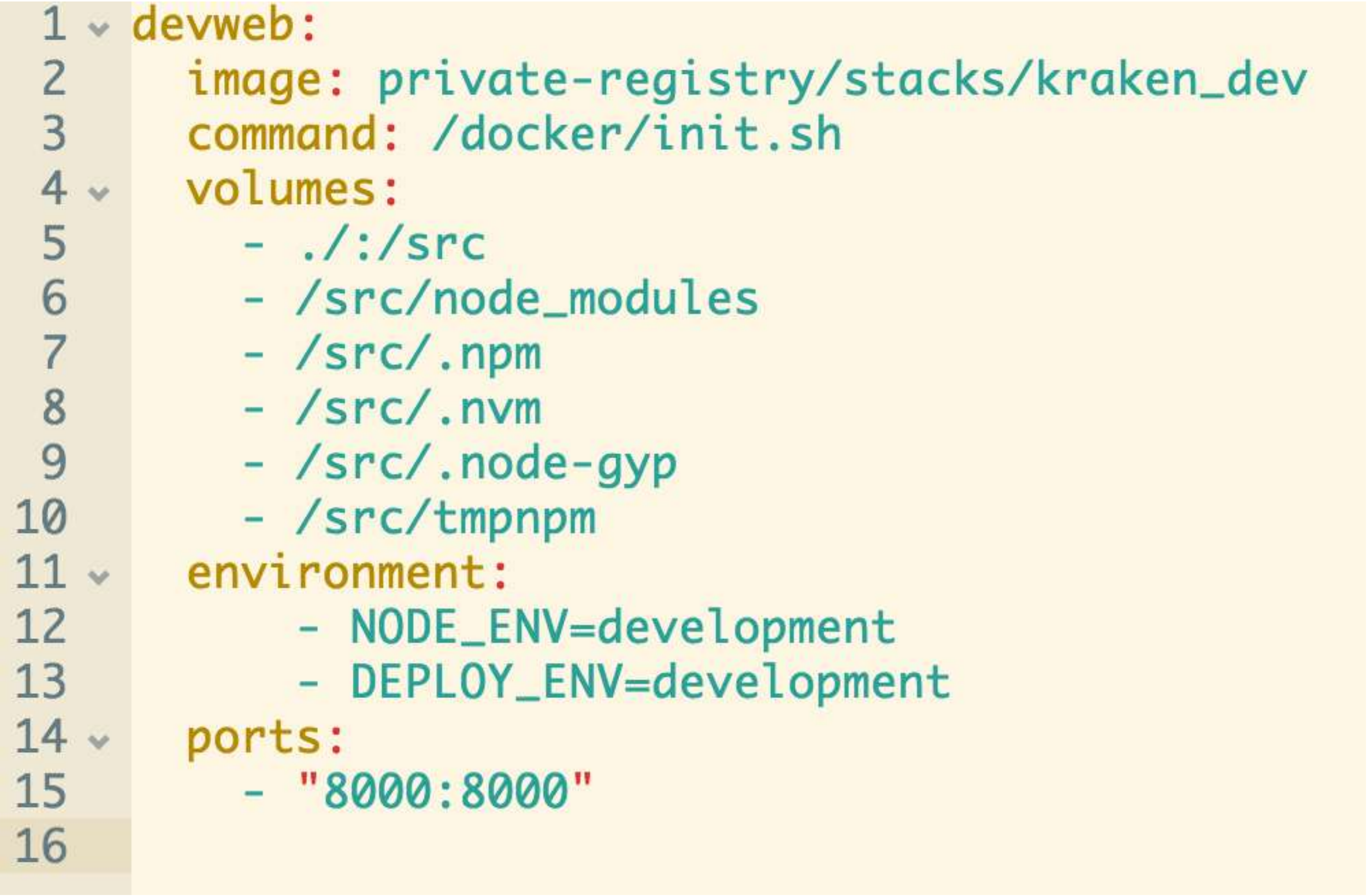
krakenjs.com

Demo repo: https://github.com/mohitsoni/dockercon15demo

Highlights:

- Running krakenjs (nodejs) application inside Docker
- Live debugging an application inside a container
- Working across multiple branches
- Simulating QA environments from laptop

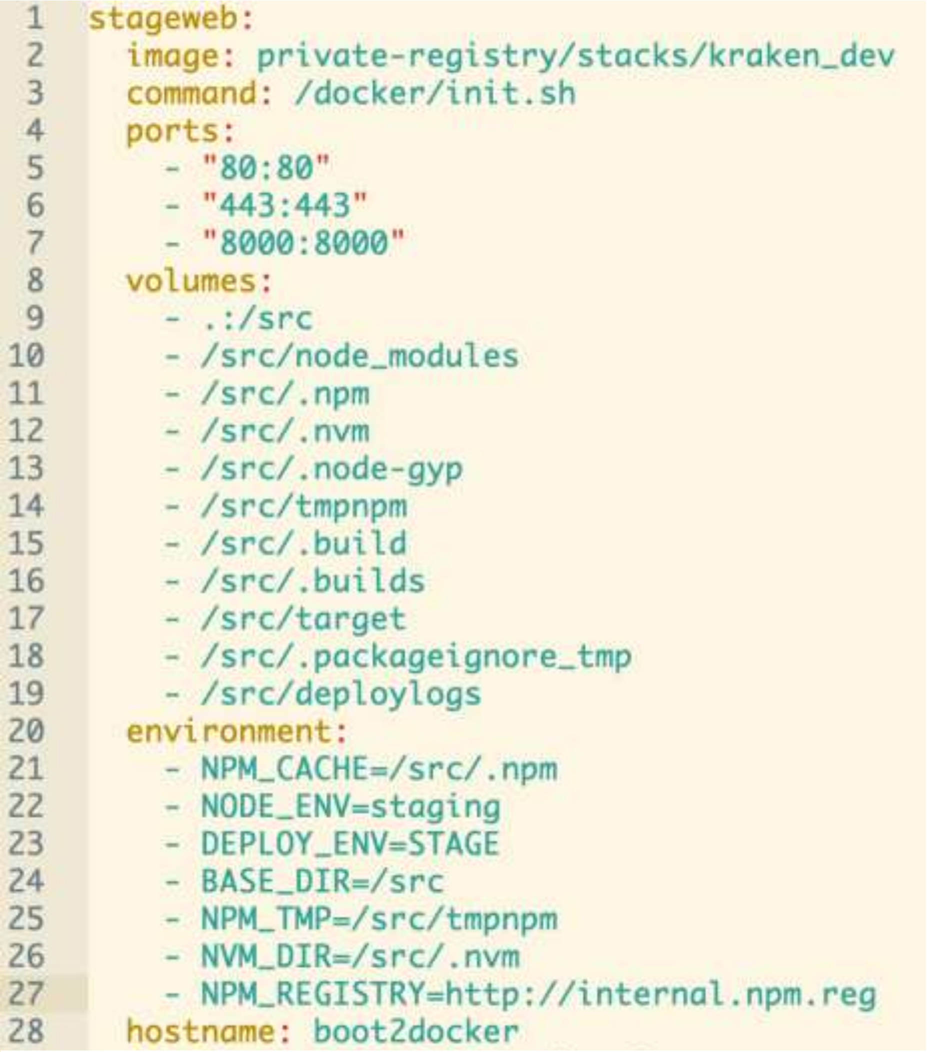
Container with Dev configuration



```yaml
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 QA configuration



```yaml
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
```

Resources

- https://github.com/mohitsoni/dockercon15demo
- https://github.com/misho-kr/elasticsearchindex
- https://github.com/misho-kr/docker-registry.all-in-one

Thank you

Mohit Soni
Software Engineer
@mohitsoni

Ashish Hunnargikar
Software Engineer
@hunnarg