Moving from Docker Swarm mode To Kubernetes

•••

Sreehari Mohan
Myntra

About Me

- Senior Systems Engineer at Myntra
- Infrastructure & Release engineering programs
- Docker, Golang, Node.js, React
- Self service tools

- https://twitter.com/sreeharimohan
- https://www.linkedin.com/in/sreeharimohan/
- https://www.facebook.com/sreehari.mohan

Myntra - 2 years ago

- 150+ services
  - Java
  - Node
  - Golang Python
- Environments had to be shared for development
- No control on who can deploy to the different environments
- What went to production?
- Unable to test features in parallel
- Environment downtime causing delays in releases to production
- No one knows how to bring 3rd party services up with the correct configurations once they went down
- Development blocked due to instability of dependent services

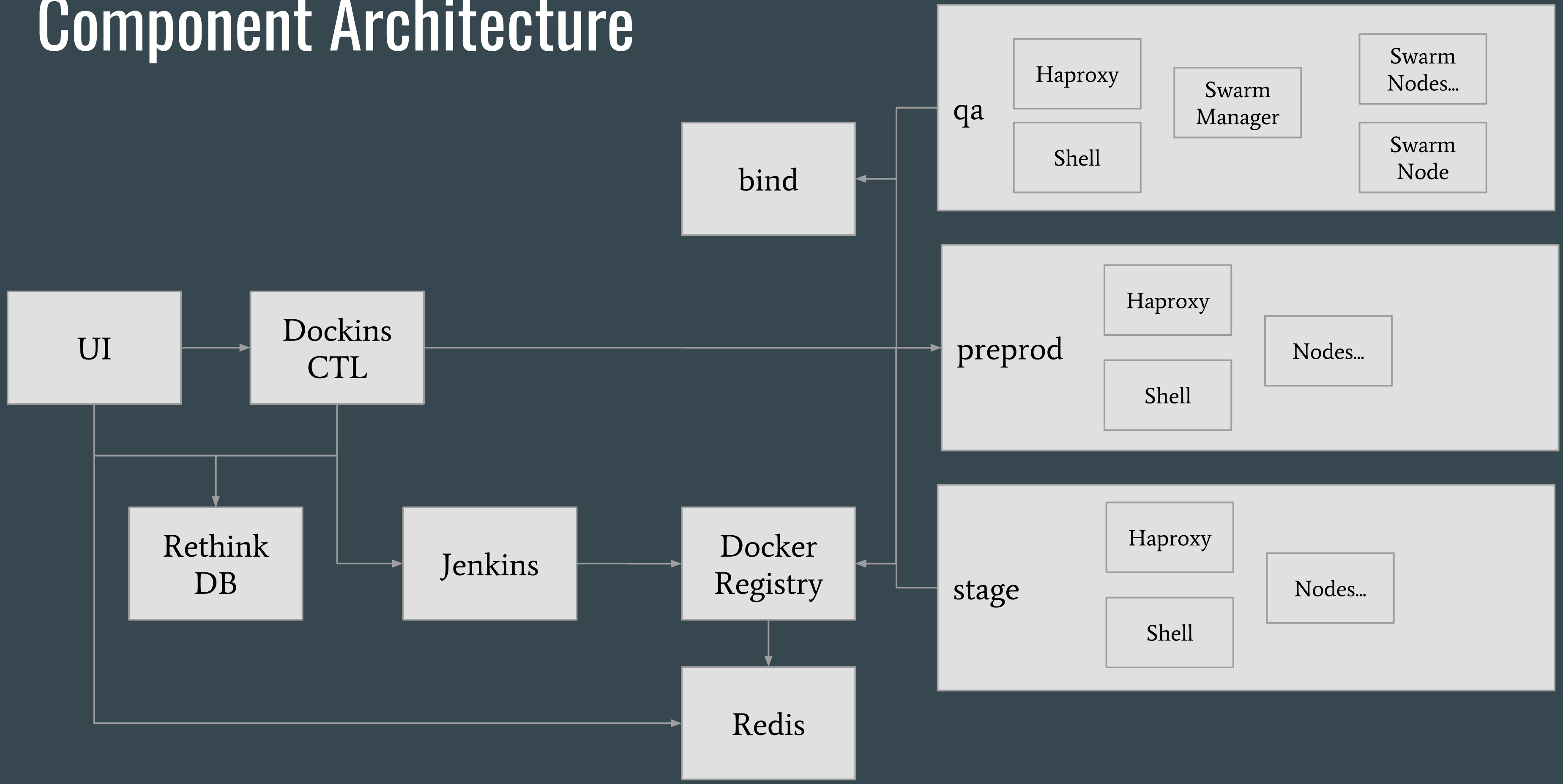
Docker was the way out!

Solution

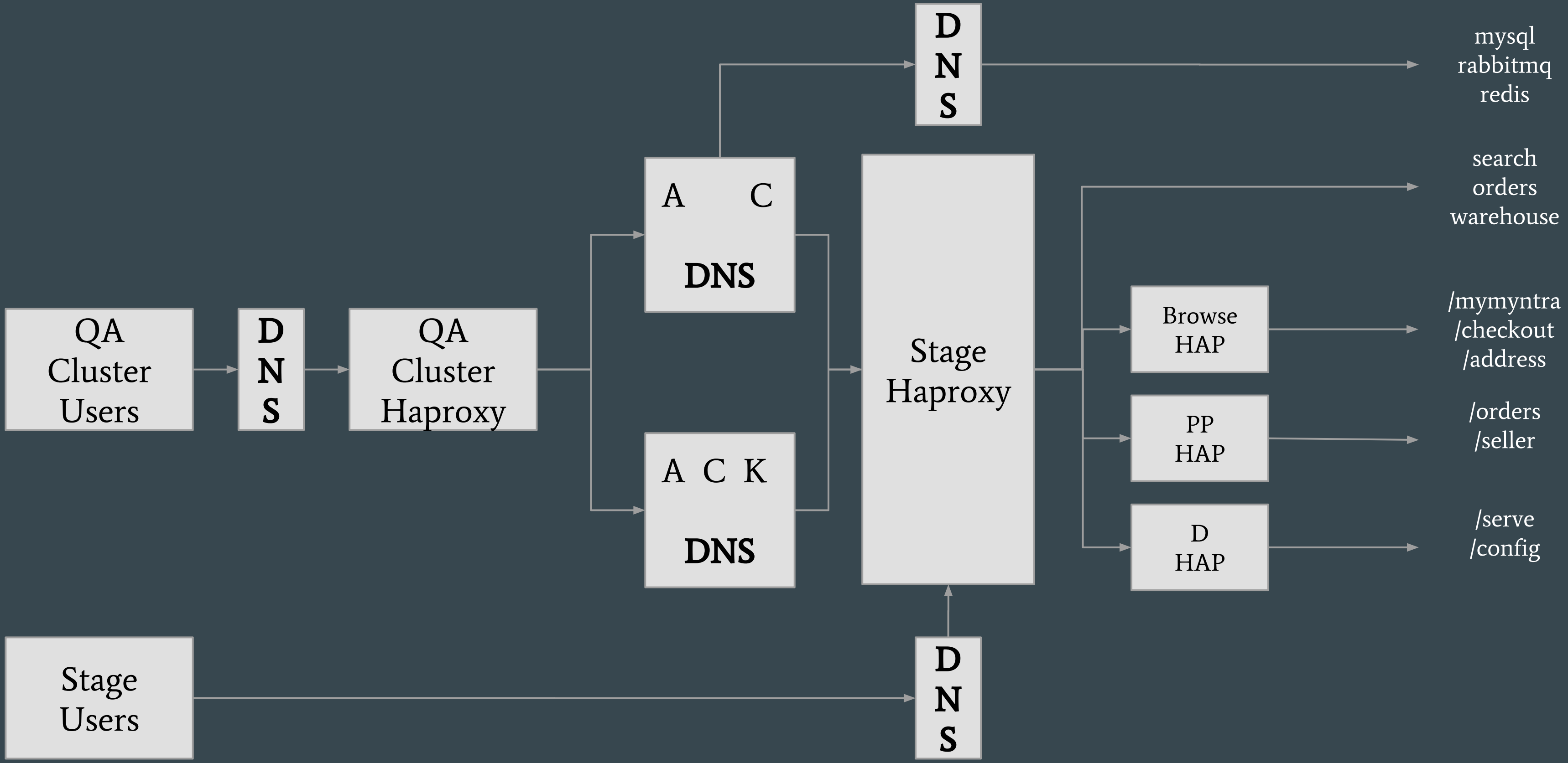
- Docker Swarm mode based sandbox environments orchestration
- Every developer could create env clusters with the services of their choice
- Network and routing automation
- Onboarding new services automation
- Environments could be brought back up since all startup configs are baked into docker images
- Multiple projects touching several services could be developed and tested simultaneously with clusters
- Clusters talk to a staging environment (prod synced) services for dependencies

Dockins is born

Component Architecture



Networking


QA Cluster Users
DNS
QA Cluster Haproxy
A C
DNS
A C K
DNS
DNS
Stage Haproxy
DNS
Stage Users
mysql
rabbitmq
redis
search
orders
warehouse
Browse HAP
/mymyntra
/checkout
/address
PP HAP
/orders
/seller
D HAP
/serve
/config

And then we needed to scale

Problems

- High number of clusters were used
- Clusters had more than 20 services in them
- Resource crunch
- Swarm mode was breaking down
- Networks were not up to date with latest services
- Routing logics were inconsistent

Docker had done really well in containerising the whole Myntra stack but was letting us down in networking

Kubernetes !

Differences

DOCKER

- Excellent containerisation model
- Swarm mode was configurable
- Only overlay networking for multi-host networking
- Uses existing network topologies and thus more complex
- Embedded DNS server on each swarm network

KUBERNETES

- Choose your own containerisation model
- Multiple configurational options
- Multiple options for multi host networking like calico, flannel, weave
- Options to use Simple IP routing thus making networking very simple
- Pod based IP maps on nodes and etcd

Convert all Dockins constructs into K8

Staging

EXISTING DOCKER SWARM

- Docker containers running with ports exposed on their hosts
- Fixed machines
- Data persistence
- Haproxy to route from 80 to specific ports for HTTP connections
- Fallback from swarm networks
- TCP based apps had to be manually added into a common DNS server

K8 MULTI NODE CLUSTER

- Namespace stage
- Fixed machines are not needed
- OpenEBS to push data along with containers as a part of the pod
- External Haproxy configurable during creation of the service
- Ingress Network Policy from all pods which serve as a part of a QA cluster for fallback
- TCP and HTTP based apps can be accessed via DNS hostname

QA

EXISTING DOCKER SWARM

- Service discovery
    - Aliases within cluster
    - Fallback DNS on host
- Network based separation to create a sandbox
- Haproxy routing from local to specific services

K8 MULTI NODE CLUSTER

- Egress Network Policy comes in 1.8
    - Namespace policy to staging
- Ingress Default-Deny
- Multiple rules together will create the sandbox
- Much stable sandbox since there is no network topology involved
- Haproxy routing will be to namespace specific urls

Thank you