Using Kubernetes for handling second screen experience of European TV show
About us

Key Qualifications

- 12+ years' experience with development and operations
- Automating everything
- Been handling Kubernetes clusters in production for 2 years

Key Qualifications

- 20+ year of experience in web-development & operation/hosting
- OpenSource advocate
- Extensive hand-on experience with Kubernetes and Containerization

Thomas Hector
Head of IT Operations

Jan-Erik Revsbech
CTO
About Peytz & Co

- 5 offices
- +100 employees
- OpenSource
- Infrastructure projects
- Custom built IT solutions
- 15 years’ experience
Danish TV Show
The task

Danish television broadcaster TV2 asked us

- To create an application which works on all devices
- To create an admin interface to run the show
- The application should be able to scale quickly but keep lowest cost possible
- Users must identify themselves (log in)
- Gametickets must be e-mailed to users
Challenge: Timing!

Weekly show - 9 weeks in a row
Attempt to get people to sign up early - unsuccessful
Hundreds of thousands users signing up during commercial break.

1 week - 1 hour
Tease before commercial break
8 min
Show-time !!
15 min
First row winner
15 min
Second row winner
20 min
Final winner
End of show

API Requests
Participants
Application architecture
Application architecture

- PHP-based Admin application
- Shared state in Redis and SQL database
- Scalable user-facing API in GoLang
- Queuing system
Application technologies

Best of breed-technologies

Right tool for the right job

Easy prototyping with php - good performance with GoLang
Scaling and orchestration

Containerized development and production environment

Cloud orchestration with Kubernetes

Hosted on AWS
Deployment, CI and CD

- Local development
- Build
- Test
- Push to dockerhub
- Deploy

GitLab
Gitlab runner
Dockerhub

Pod
Pod
Pod

peytz&co
Application infrastructure
The services around Kubernetes

Experience

Take control

Easy to manage and scale

aws
Amazon ElastiCache
Amazon RDS
kops
KOPS: Installation

+ Create AWS user
+ Make a CI host
+ wget and install kops and kubectl
+ Configure AWS cli
+ Create S3 Bucket
+ Export KOPS variables
+ Setup Route53
KOPS: Create cluster

root@ci:~# kops create cluster $NAME \
--zones=eu-west-1a \
--dns=private \
--authorization RBAC \
--master-size m3.medium \
--node-size t2.medium \
--yes
KOPS: Edit ig nodes

```yaml
apiVersion: kops/v1alpha2
kind: InstanceGroup
metadata:
  creationTimestamp: 2018-09-02T09:36:58Z
labels:
  kops.k8s.io/cluster: k8s-cluster.shanghai-k8s.vpc
name: nodes
spec:
  additionalSecurityGroups:
  - sg-1e288
  image: kope.io/k8s-1.8-debian-jessie-amd64-hvm-ebs-2018-01-14
  machineType: t2.medium
  maxSize: 3
  minSize: 3
  nodeLabels:
    kops.k8s.io/instancegroup: nodes
  role: Node
  subnets:
    - eu-west-1a
```
KOPS: Edit ig nodes

```yaml
apiVersion: kops/v1alpha2
kind: InstanceGroup
metadata:
  creationTimestamp: 2018-09-02T09:36:58Z
  labels:
    kops.k8s.io/cluster: k8s-cluster.shanghai-k8s.vpc
  name: nodes
spec:
  additionalSecurityGroups:
  - sg-1e288
  image: kope.io/k8s-1.8-debian-jessie-amd64-hvm-ebs-2018-01-14
  machineType: m4.xlarge
  maxSize: 20
  minSize: 16
  nodeLabels:
    kops.k8s.io/instancegroup: nodes
  role: Node
  subnets:
  - eu-west-1a
```
### KOPS: Validate cluster

#### INSTANCE GROUPS

<table>
<thead>
<tr>
<th>NAME</th>
<th>ROLE</th>
<th>MACHINETYPE</th>
<th>MIN</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>Node</td>
<td>m3.medium</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>master</td>
<td>Master</td>
<td>m4.large</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>monitor</td>
<td>Node</td>
<td>m4.large</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>nodes</td>
<td>Node</td>
<td>m4.xlarge</td>
<td>16</td>
<td>20</td>
</tr>
</tbody>
</table>

#### NODE STATUS

<table>
<thead>
<tr>
<th>NAME</th>
<th>ROLE</th>
<th>READY</th>
</tr>
</thead>
<tbody>
<tr>
<td>ip-172-20-50-127.eu-west-1.compute.internal</td>
<td>master</td>
<td>True</td>
</tr>
<tr>
<td>ip-172-20-50-144.eu-west-1.compute.internal</td>
<td>node</td>
<td>True</td>
</tr>
<tr>
<td>ip-172-20-50-170.eu-west-1.compute.internal</td>
<td>node</td>
<td>True</td>
</tr>
<tr>
<td>ip-172-20-50-179.eu-west-1.compute.internal</td>
<td>node</td>
<td>True</td>
</tr>
<tr>
<td>ip-172-20-50-37.eu-west-1.compute.internal</td>
<td>node</td>
<td>True</td>
</tr>
<tr>
<td>ip-172-20-50-53.eu-west-1.compute.internal</td>
<td>node</td>
<td>True</td>
</tr>
</tbody>
</table>
Monitoring

Prometheus autodiscovery

Grafana dashboards

Graylogs search and dashboards
Performance test: LOCUST

- Test as code
- Highly scalable
- Less hardware consuming
LOCUST: Fast Forward

- Connections
- nsdc
- RDS IOPS
- 1U
- Cluster
- Firewall
- 3 Cluster 3 Data Centers
- Pods HPA
- ELB
- RDS CPU
Evaluation & conclusions
Evaluation

Kubernetes and docker for fast scaling

Low price tag on operations

No bottlenecks throughout the season
Record breaking
Second screen interaction

62%
Conclusion

- Documentation and procedures
- Prometheus
- Scaling
- Best practice
- Locust
Thank you for the attention

Any questions?

Thomas Hector
th@peytz.dk
@thomas-rasch-ector

Jan-Erik Reysbech
jer@peytz.dk
@janerikrrevsbech