hotstar tech_

Hotstar's journey from EC2 to Kubernetes

CHALLENGES & LEARNINGS

This document consists the walkthrough from ipl 2018 to 2019.

About Hotstar

- #1 OTT platform in India
- Over 350M+ downloads
- 1 Day,100M users, 2.5X increase in concurrency
- Available in >15 languages
- Variety of content
  - Live/On-Demand
  - Sports/News/TV/Movies
  - Regional Catalogue

hotstar tech_

Let's talk about Scale!

25Mn+
PEAK CONCURRENCY DURING WC Semifinal

1Mn
PEAK REQUESTS PER SECOND

10 Tbps+
PEAK BANDWIDTH CONSUMPTION

10Bn+
CLICKSTREAM MESSAGES

100+
HOURS OF LIVE TRANSCODING EVERYDAY

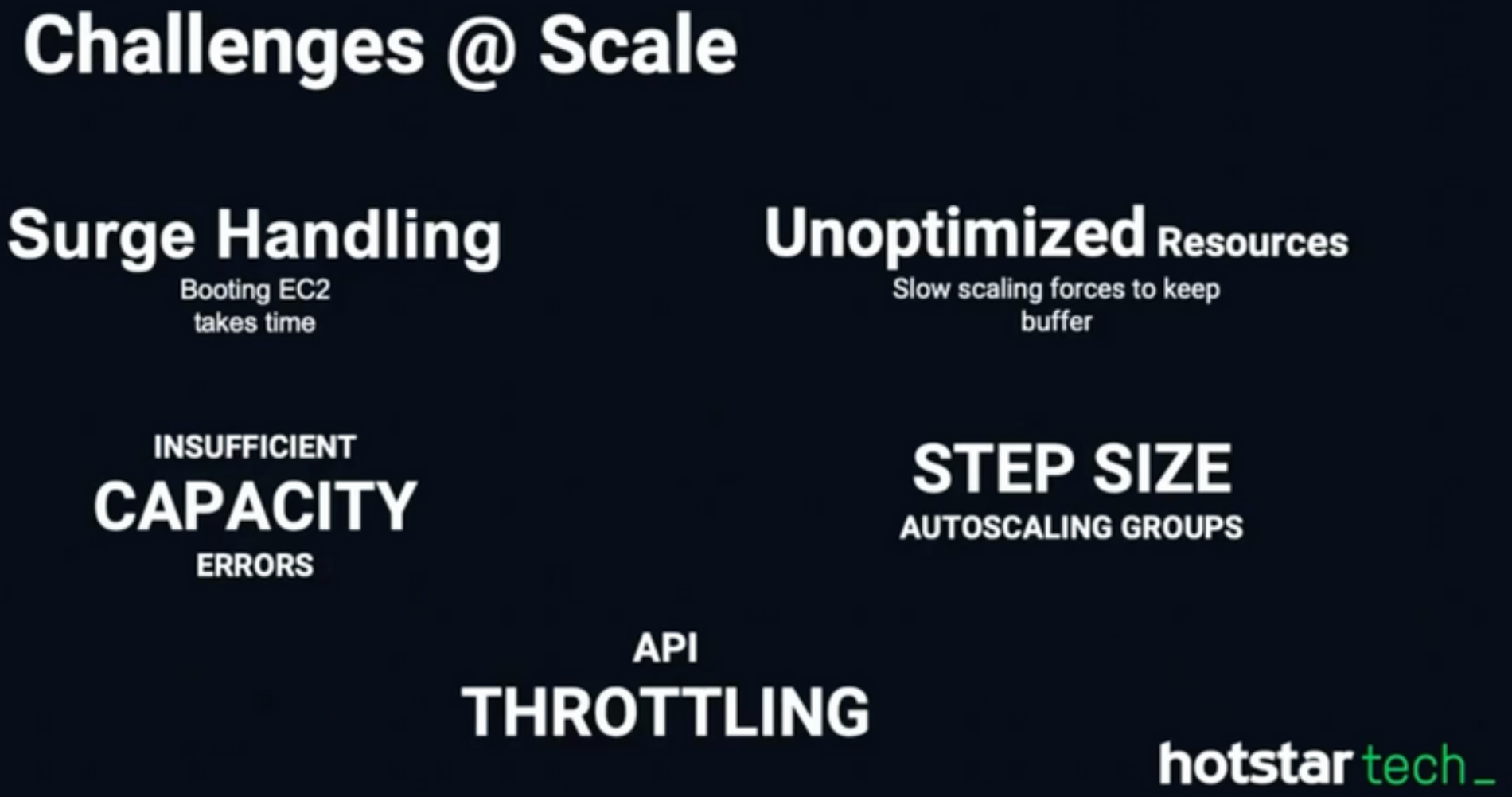
hotstar tech_

Following pic depicts journey of Hotstar when they started at 2018:

They had ELBs which were like ingress to them and they had them attached with ASGs which scales the application as per the requirement.

They used AMIs and python for the deployment at EC2 machines.

They used Terraform for infrastructure deployment and asked everyone to not do any manual changes because whenever the automation scripts executes or deployment goes on, existing infra will be modified and production issues may arise.



When something interesting happens during the game, a surge of users comes into the platform and they need to handle that seamlessly.

But due to the boot time of EC2 they were not able to scale as expected.

They used to keep buffer of the resources as they knew that at any time there can be a surge.

But having buffer with very large scale is not good as it can lead to hardware issues and capacity issues in case the AZ's are not available.

EC2 itself calls AWS api's internally and they had seen throttling of those api's which enforced them to do step scaling instead of scaling all at a time. They also did account level rate limiting.

They realised that they are missing something with their existing logic.

One of the plus point of container is that we can run them anywhere.

Local deployment becomes fast with the help of config files.

As compared to EC2, it's quite easy to spin up a new container.

Resources are optimised because they earlier used to have whole EC2 machine for one application.

Let's say their JVM is running, it requires 2.5 cores and some x amount of memory.

The lowest CPU they are getting as a hardware is 3 core or 4 cores. Then they had to take 4 core and can't deal with 2 core machine.

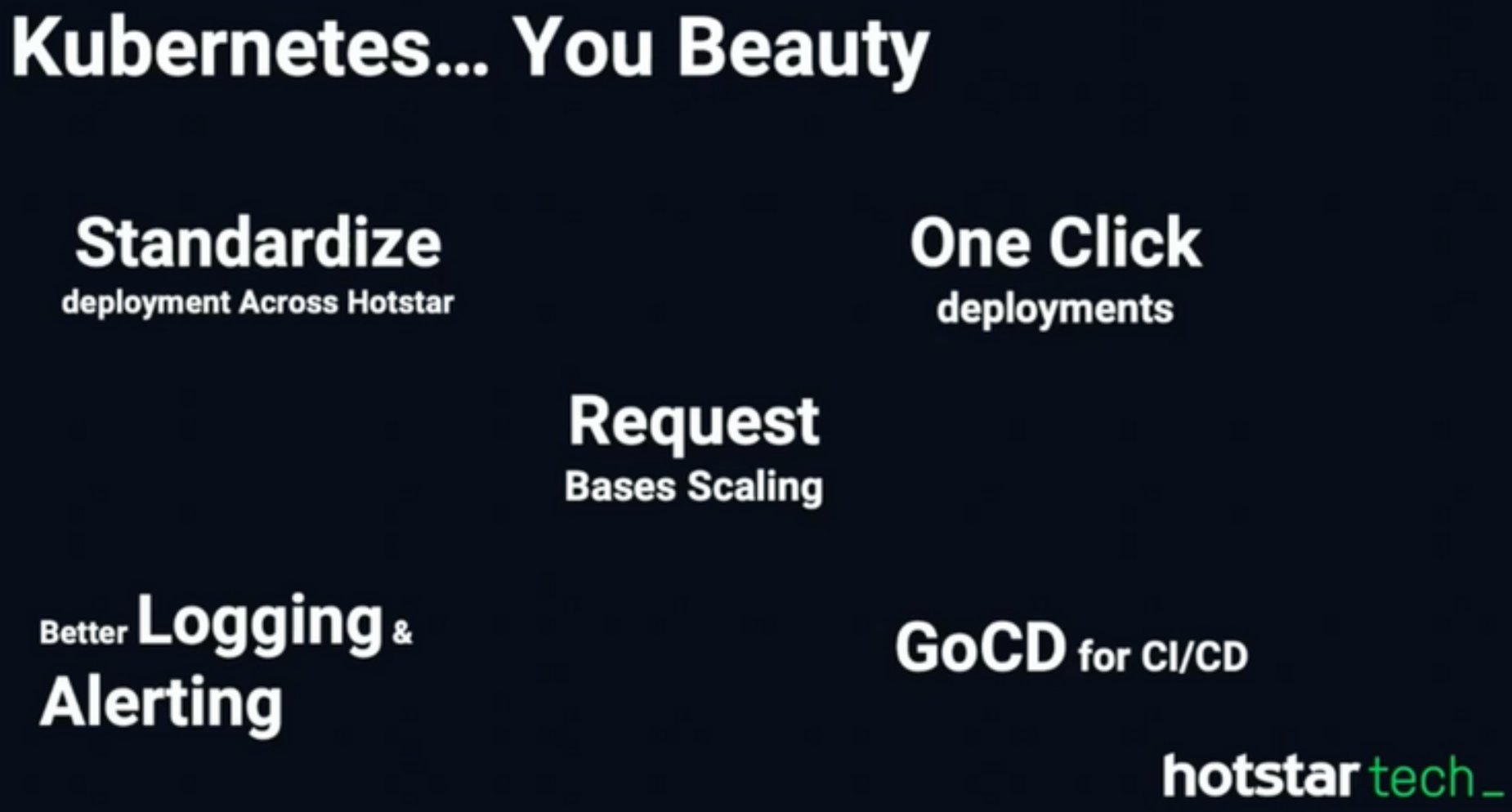
These are some issues they realised that at scale , they can bite when the resources aren't available.

With docker we can specify whatever the memory the container wants.

As the boot up is easy, scaling becomes seamless and we can scale it to any number in comparison to EC2.

The whole deployment is standardised, earlier some were deployed in their own way and others were using different instance group.

Debugging became easier for them because of this.



They convert JSON to K8s yaml with the help of Jsonnet then apply it to the cluster

Logging was standardised, earlier different kind of logs were produced at different applications .

They use GoCD for CI/CD and its deployed at k8s cluster.

The best thing they found is the request base scaling for their application. Earlier in EC2 world, they used to get the concurrency and they had written scripts for scaling upto 1.5x to 2x.

They manually used to create the scripts, perform scaling manually and test them later on.

Application became intelligent themselves to scale. They spin up the containers based on the horizontal pod scaling which k8s provides and they're getting metrics from prometheus.

Earlier someone is responsible for scaling the Hotstar but now it will scale on its own.

Introducing new things for a team is okay but for a organisation , lot of challenges needs to be faced.

To dockerize a application, developers can use Dockerfile and create containers or they can build applications on top of based docker images.

It can't take GBs to download the base image.

Earlier applications were running on single EC2 machine, whatever memory they want they were getting.

Application performance they had to check and its like changes are going first time to the production.

With the features, they were getting they optimised the cost and it was good for them.

They had to give the number to k8s that if they are getting 10k requests and they should scale.

Finding that number was difficult for them.

They manage their own cluster cluster management v2.

With the scaling of application, infra components should also scale like prometheus and file beat.

The DNS they are running inside cluster they need to ensure they are doing it correctly.

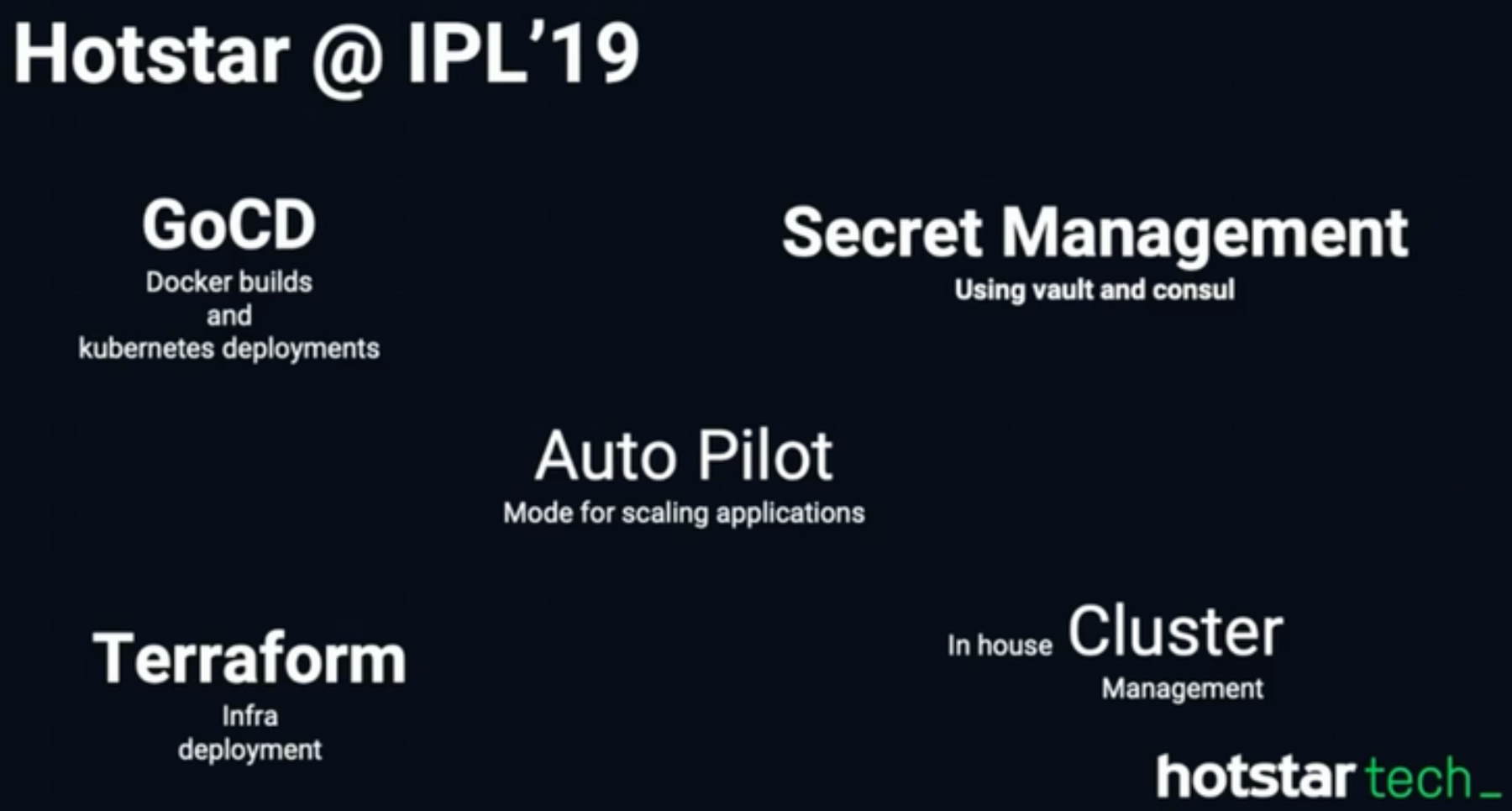
They used to keep platform ready for the game day.

They had to scale the nodes

They had to tune the scaling parameters for load testing and scale better.

If the infra resources are not scaling it can affect at cluster level whereas application can only affect one part.

They had to scale the nodes



At runtime they used to inject secrets using vault at runtime to docker containers.

They love terraform and used that for infra deployment.

They were doing in-house cluster management like some of the core components of infra.

Auto Pilot was one of the main feature they achieved during the migration. I case of EC2, they had to keep a buffer of resources and they were wasting a lot of resources there. They had to talk with AWS for that.

EC2

- Limitation of request handling at load for services.
- Pre-scaling leads to up-optimized resources.
- Patching infra components were difficult as they were distributed.
- Scaling through ASG needs manual efforts during live events.
- Each application handled it own monitoring and alerting .

K8s

- Application scales as per the load based on requests processing.
- Applications scales based on traffic it receives.
- Patching base docker images reflects across all the applications.
- Auto Pilot mode because of request based scaling for applications.
- Centralised monitoring and altering with standard deployments.

They were not able to customise AWS ECS as per their requirement that time. Earlier each application was using EC2 and they were wasting lot of resources but k8s helped them to focus on the core components.

Summary

Scaling
Application
Better in k8s as
compared to ec2

EC2 scaling leads to keep
extra resource for applications

Auto Pilot mode during
live events

Less go to **Production**
time for applications

Standard **deployment**
strategy across applications