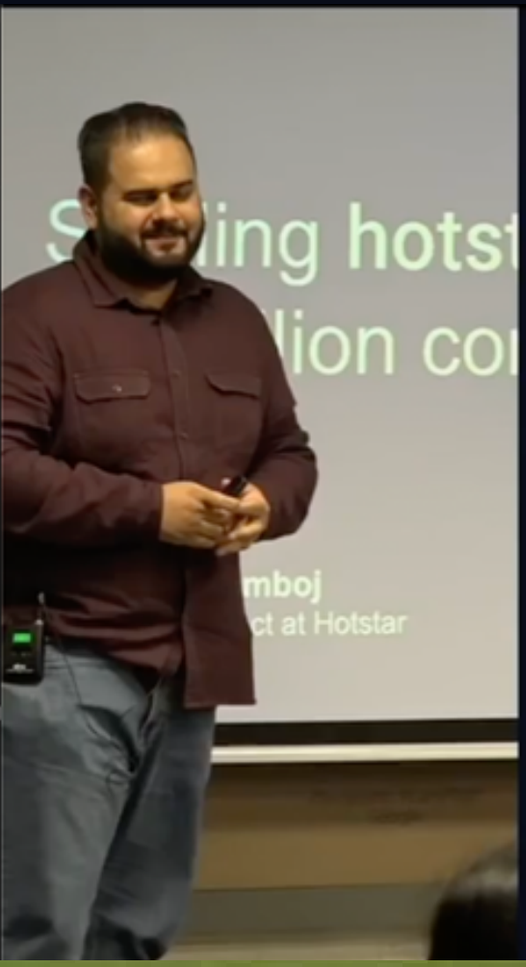
Why Hotstar doesn't makes use of auto scaling groups
for live streaming and what are clickstream messages,
step size ASGs, tsunami tests, spot fleets,
game of availability zones,
project Hulk load testers, chaos engineering
and panic mode?

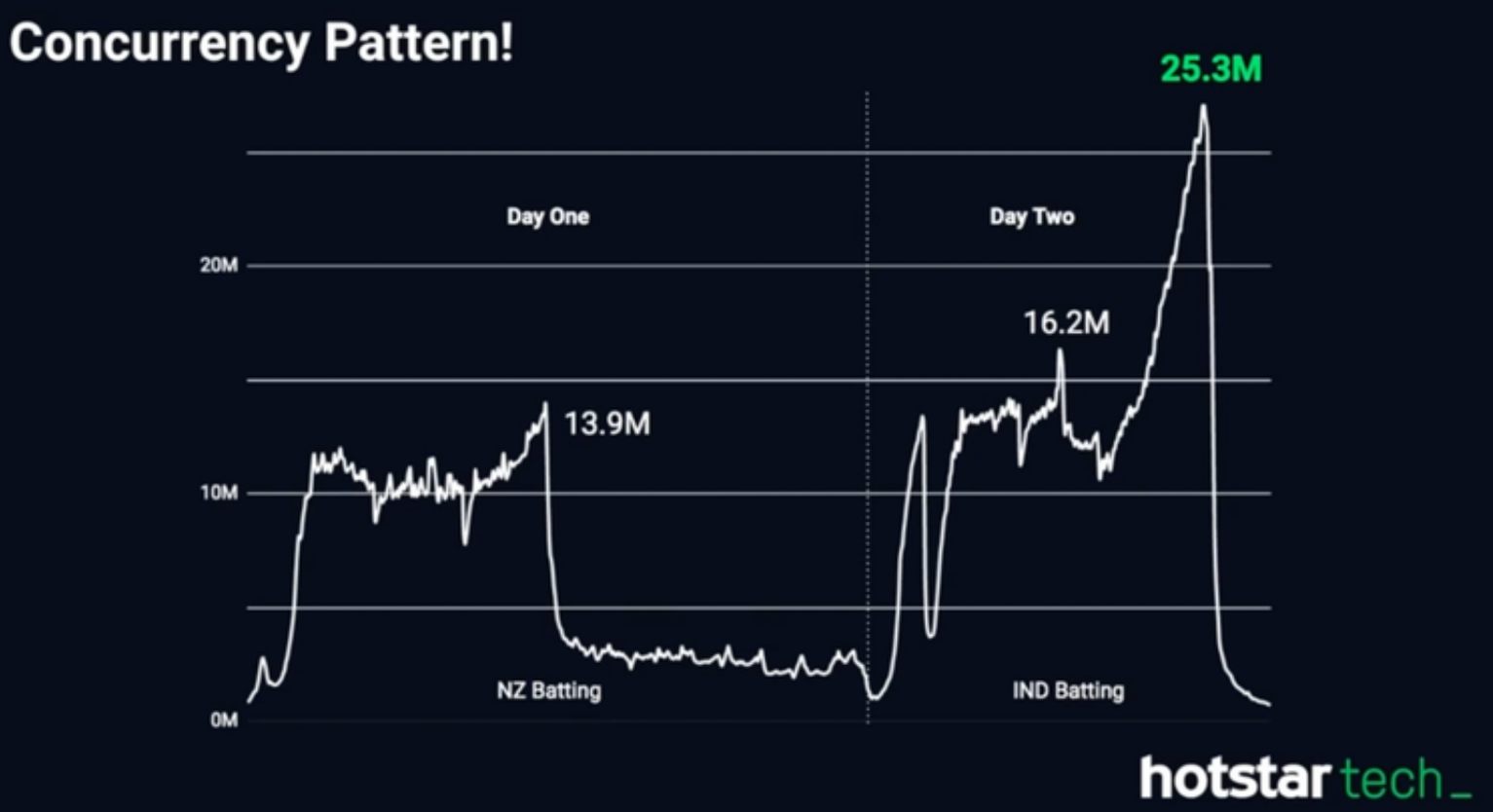


hotstar tech_

Scaling hotstar for 25 million concurrent viewers

Gaurav Kamboj
Cloud Architect at Hotstar
@oyehooye

Gaurav manages infrastructure , security and monitoring at hotstar



Toss happened around 2:30 , and usually 3-4 million come at that time and the initial spike represents that.

And there's an 30 mins break and the second spike represents the start of the game.

The game was played over a duration of 2 days and it was rain affected.

Most of the time the traffic is around 10 million and that shows how stable the platform is.

Then wickets kept on falling in regular intervals

But suddenly when Dhoni came on, their marketing team started sending push notifications.

The spike increased afterwards upto 25.3M . It's like 1.1M users added to the platform per minute.

At that point dhoni got out and the spike came down quickly. That's the time hotstar made a global record previously.

Suddenly , when users switch from video to homepage. They make home screen calls.

They have the sections at homepage like personalized content and what one watched previously.

When all the 25M users come to the homepage back , it needs to handle that load.

That's what they had to prepare, they don't know when the traffic will drop and what's the peak will be.

The spike represents drinks break which usually happens around 16th over and 32nd over.

NZ batted first and they went upto 13.9M.

When the rain started , there was a sudden decline upto 5M.

But they still waited upto 3 to 4 hrs for the match to begin again.

Then the day ended but people were so eager to watch that match again

Then they came to bat the next day with 4 hrs .

Quickly then India padded up and came to start their innings.

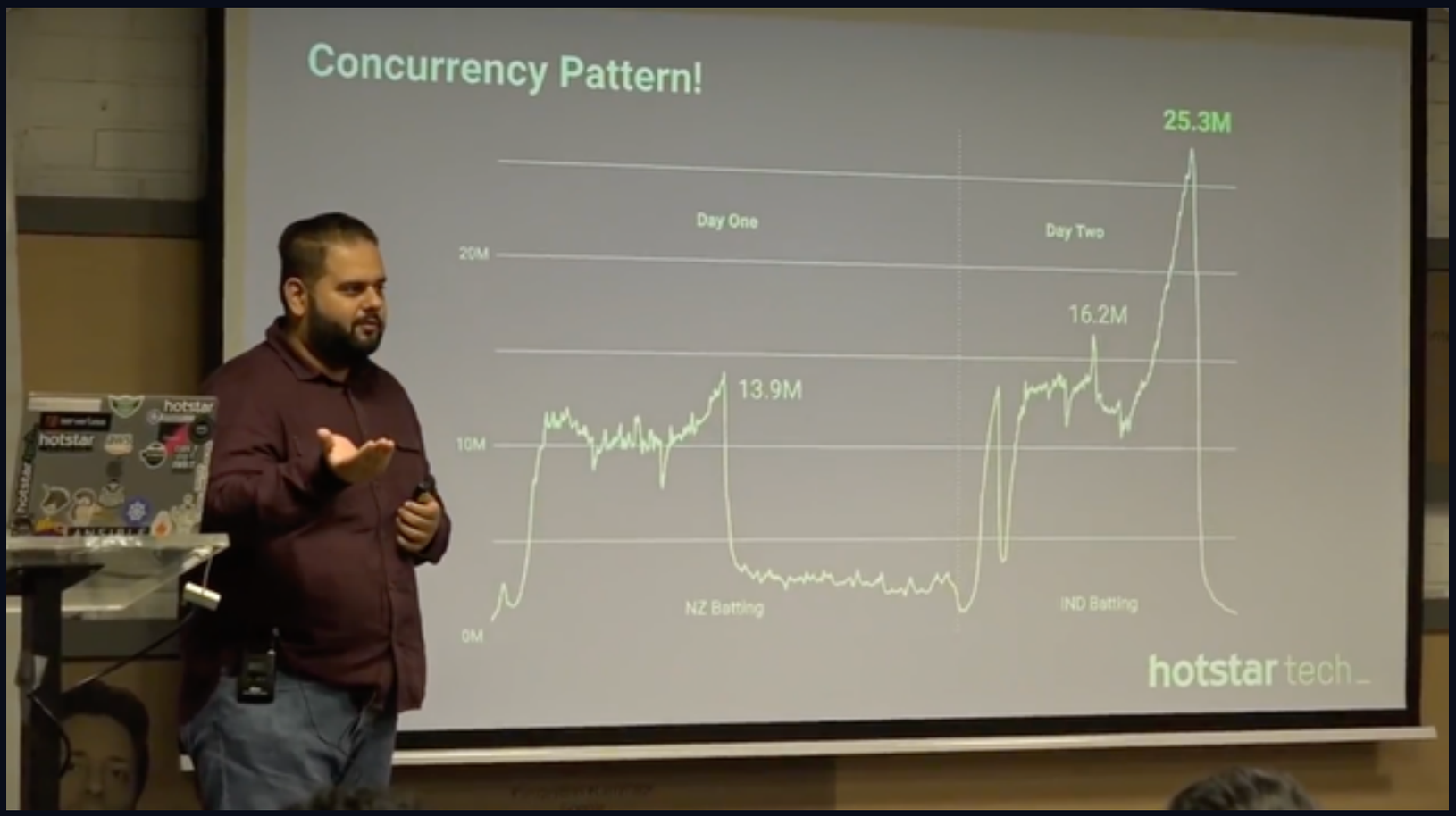
This was the most critical part for hotstar's team.

From the start of second day , spike went from about 1.5M to 15M and then there was a sudden drop.

And this is very harmful for the backend services. All the services need to be scaled up well in advance .

We can't rely on auto scaling since it's very slow in nature.

And the entire scale up on the second day was about 1M per min till the rain started. That's the growth rate.



About **hotstar**

- **Disney** owned **#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_

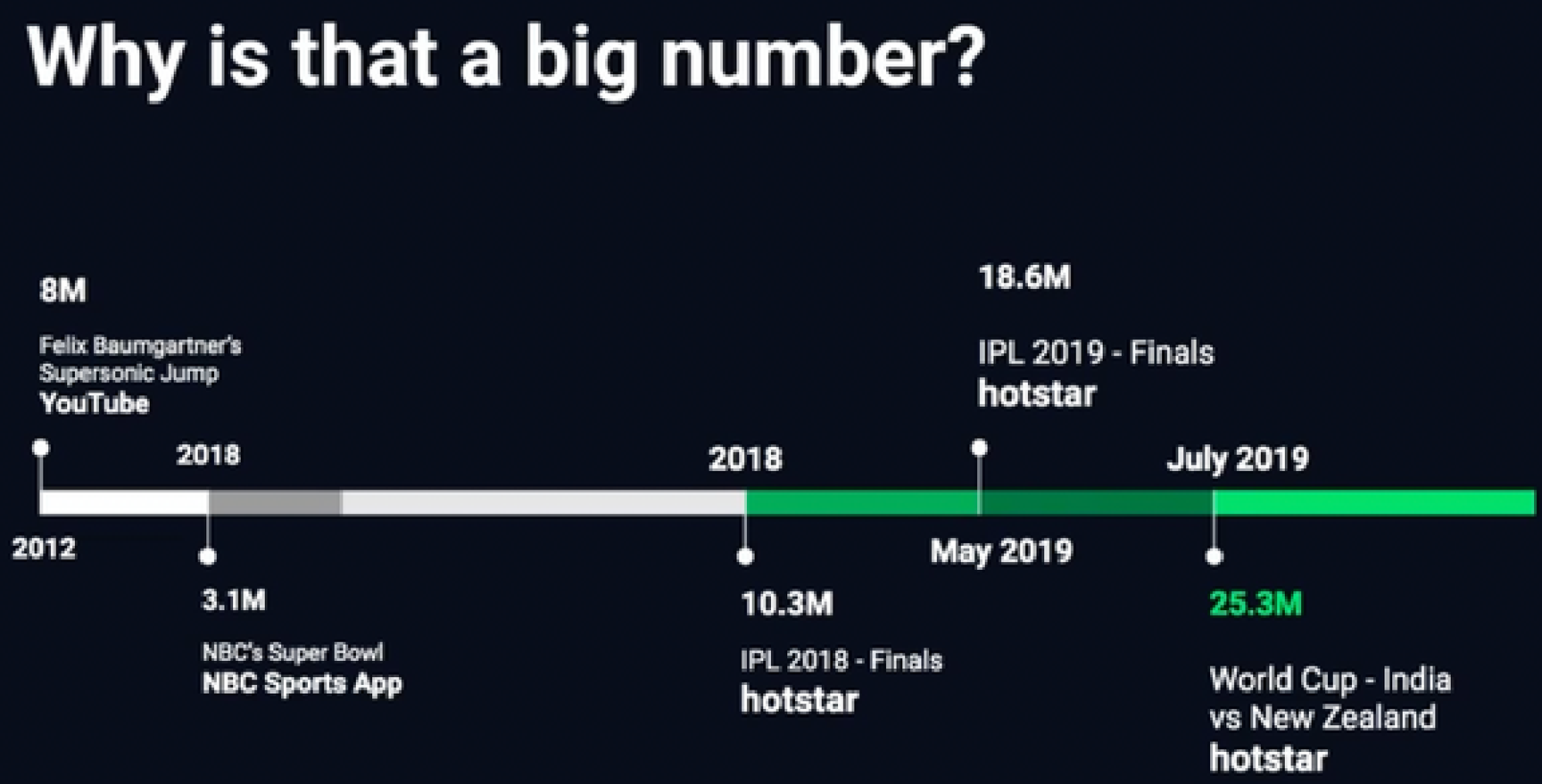
At that time, concurrency has grown 2.5x as compared to previous ipl.

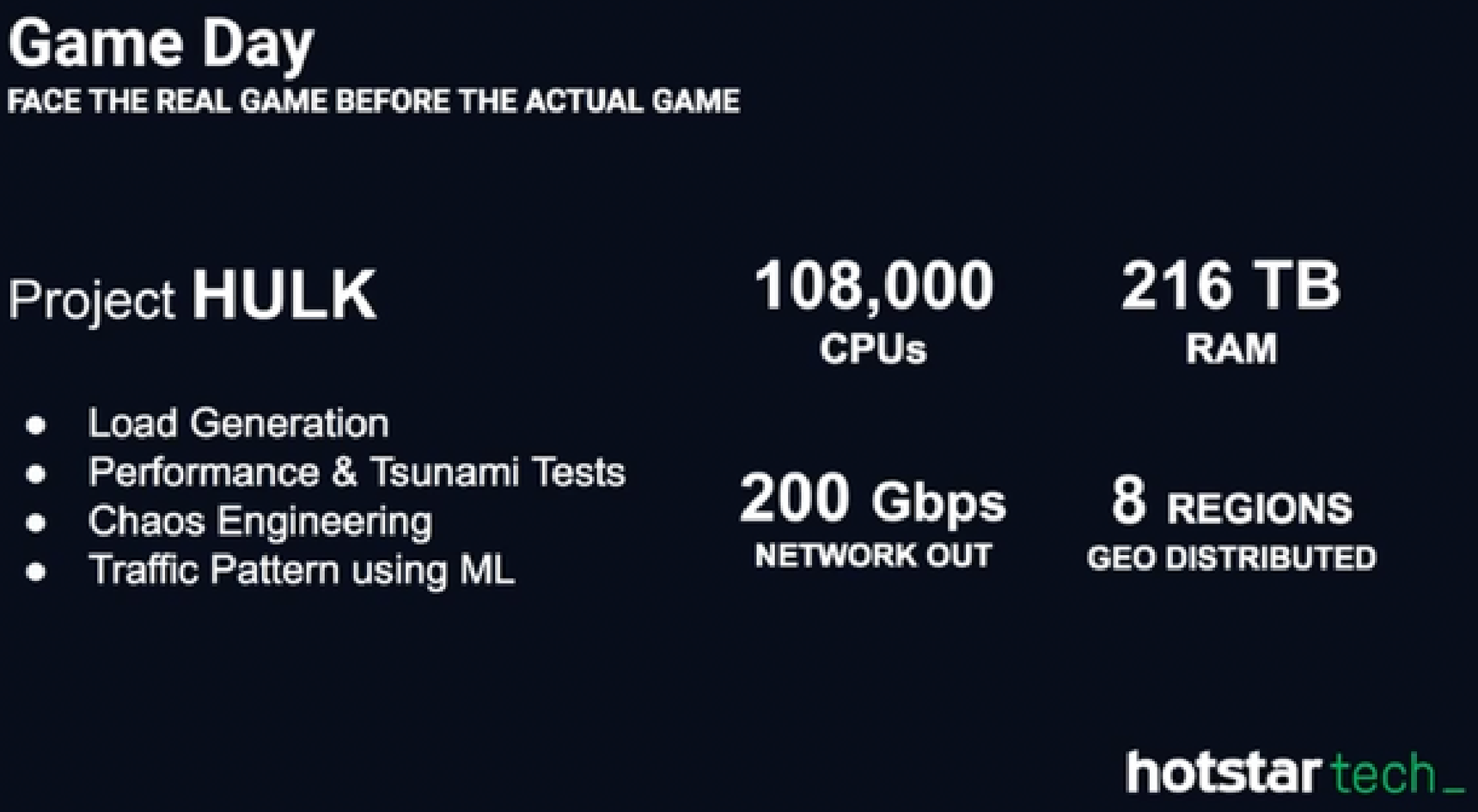
Bandwidth consumption involves utilizing a network's transmission capacity for forwarding data packets from source to destination computer. Typically, the total bandwidth harnessed in a network at any given point is less than its theoretical maximum data transmission rate.

A clickstream is a record that contains data about a website user's clicks on a computer display screen via a mouse or touchpad

(metrics which app sends to the backend)

10Tbps is about 70 percent of total internet bandwidth available in India.





It's not like auto scaling can save to handle such events

They do a lot of game days

N/W and application load tests (how much each application can handle) are performed

Hotstar has in house project called projet hulk. This is the amount of infrastructure that goes behind that load testing.

They've about 3000 or more c5.9xlarge machines just to generate the load.

Each c5 9x large machine has 36 CPU's. Each machine has a 2GB RAM if we multiply that with 3000 we'll get 10800 CPU

And this load generator will later hit their api services or applications that we saw on the first graph because they have to prepare the platform for those spikes.

200 Gbps Network Out will be generated due to the load Gen machine.

Funny part was whenever they had to do load testing, other customers were impacted.

Because in a public cloud environment they share the n/w with all the customers or the CDN partners, edge locations used to get overwhelmed.

To prevent others customers getting impacted from load testing, they moved to Geo Distributed Load Generation.

So instead of a single region , they've their load generation machines in 8 different Geo regions and all of them generates the load together.

In this way not a single location is overwhelmed when they perform their load testing.

Apart from load testing, they also do tsunami testing and its the graph they saw in the first image, the sudden surge and the tip.

That kind of spike and drop can kill any application unless we're prepared for.

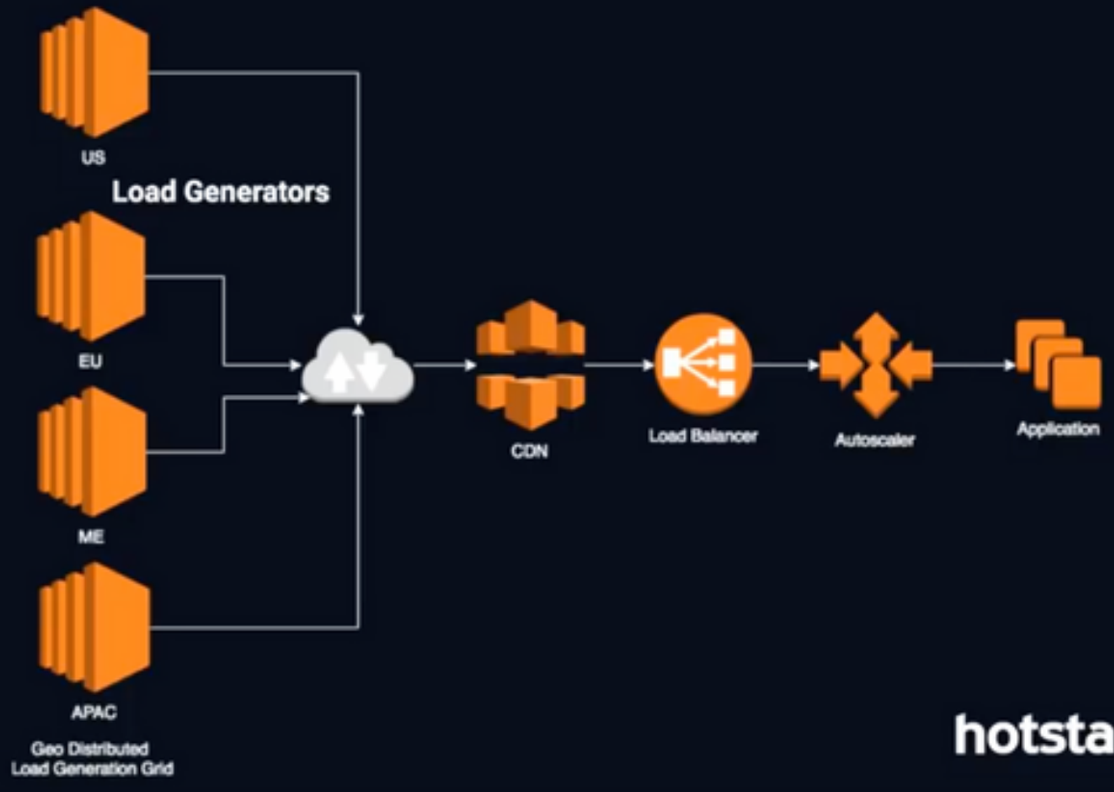
It also helps them to do Chaos engineering

With all the available information, at what concurrency each application how load each application was handling.

This is what they go back to drawing board and figure out breaking point of each system.

This helps them to figure out traffic pattern, and figure out what will happen when India bats first or 2 fav. teams are played against each other.

We get all those answers by analyzing the raw data.

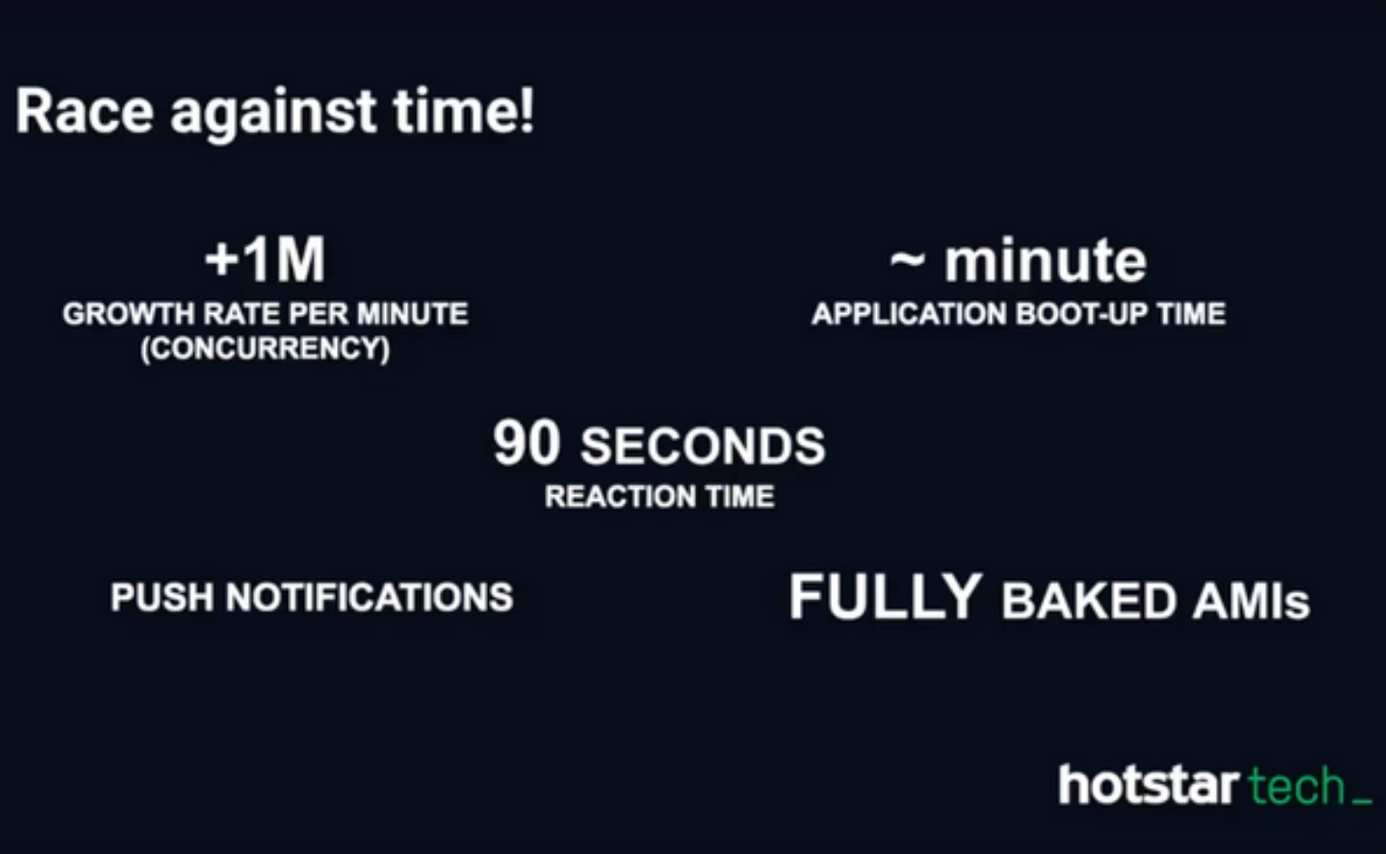


c5 9x machines distributed in 8 machines goes to CDN and load balancer (ALB or ELB)

Each load balancer has a limit or the peak upto which it can handle. We actually need to shard load balancer, for each single application they use 4 or 5 load balancers and control them via weighted routing.

Load is distributed and they were able to scale the application.

Application can either be hosted on EC2 or k8s.



Growth Rate is 1 million per minute. We can't start by 10 million/ s and start scaling up because by the time ec2 is provisioned, it boots up then application becomes healthy, register itself under a load balancer 5-7 mins are wasted, in a live match we can't afford that.

5-6 minutes traffic an increase by 5 or 6 million and that's we need to keep scaling proactively and maintain a buffer while scaling up.

The application boot time is about 1min and 90s reaction time they've to decide whether to scale up now or later or to wait.

There's a strategic timeout scheduled and we know that the traffic is going to drop, they've some reaction time there.

Marketing teams handles push notifications specially if there are interesting things happening out like Dhoni hits 3 sixes or Bumrah takes a hatrick.

These push notifications goes out to a user base of 150 to 200M users.

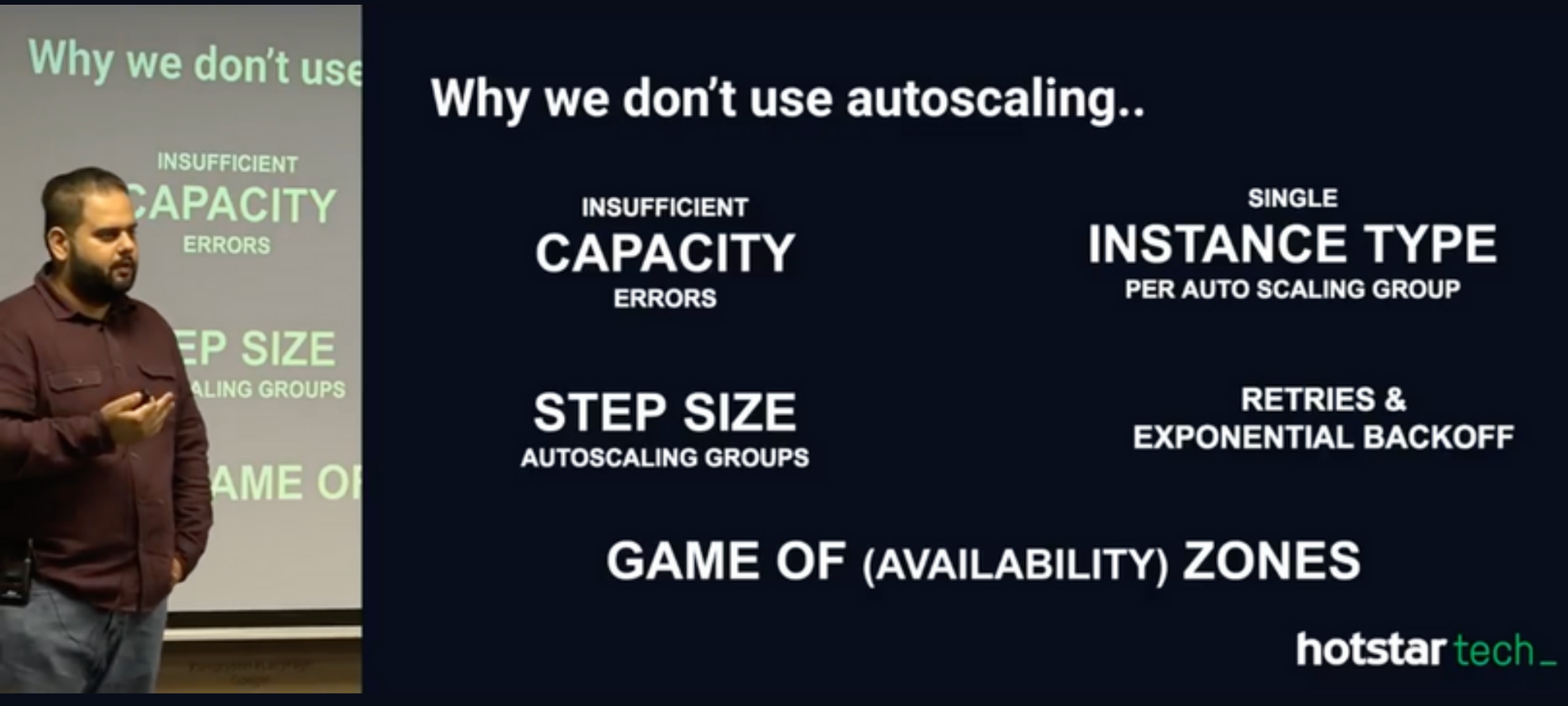
2 to 6 conversations means 4 to 6M users getting added to the platform in a very short span of time.

These push notifications can go out in any time and there can be a interesting moment anytime during the game so they need to handle the buffer as well as the spike.

Fully Baked AMI

They use only fully baked AMI and not the configuration tools because the tools like Puppet because unwanted configuration can add delay to the application from becoming healthy.

To save time we use fully baked AMIs , even the container images so that it don't have to wait for any ansible script or configure the application to make it self healthy.



Capacity is not available in a particular AZ.
When we want to go from 10M to 15M, we need to request a lot of servers.

Let's say we're operating at 400 and we need 600 servers, 200 needs to be added what if we only gets 50. Those kind of problems are there.

Let's say we're operating at 400 and we need 600 servers, 200 needs to be added what if we only gets 50. Those kind of problems are there.

A single ASG can only have one launch configuration and it will only have one instance type so if c4 4x isn't available and gives error, we can't scale the application.

Since ASG only provides one instance type we can't scale another instance if required, we are blocked here.

STEP SIZE will only happen at scale.

when we increase target or desired capacity in a ASG. It adds server in a step size of 10, 20.

Let's say we scale up application from 100 to 800, it will try to add 10 or 20 servers in all available region.

This process is very slow, we say 10 servers are added in every 10 s, it will take about 12 mins for 700 servers to add which is simply not acceptable when someone is running a live game.

And there's lot of api throttling in these cases, we can ask aws to increase the service limit or step size to scale, let's say 100 but we're doing more damage to the system in this way.

200 server launch in one go we're making 200 server calls as well and these are multiple calls like ec2 api calls, lb calls , monitoring calls to cloud watch, disk attachment api calls etc.

All happens in background and it uses all control play api which is transparent to user .At this scale there's a fix limit above which we cannot go.

Game of (Availability) Zones:

At hotstar people are fan of Game of thrones.

Let's say we've 3 AZs : 1A, 1B and 1C. What happens is once we've less capacity and we increase the target capacity of ASG.

AWS with its internal algo it has will try launching servers in all three AZs

it will launch 10 , 10 & 10 which will be successful.

Let's say in a AZ we've only 10 servers left, in the second attempt.

10 in 1A and 1B but in 1C it won't be able to launch.

If we fail to add servers to a AZ, application follows exponential backoff, like 5s, 30s, 1m, 5m, etc. It goes on increasing.

Once infrastructure becomes queue, it will harm our scaling

load servers in 1A, 1B and IC has only 10 servers. If in 1B , AZ goes down.

All the traffic will be served through 1A.

It doesn't have enough capacity to handle all the load because ideally it should be handled by 3AZs but all the load come upto 1AZ.

Second big problem is even if C doesn't have capability, ASG is still trying to launch servers there increasing the scaling time.

If one need to move from 100 to 800, instead of getting the servers provisioned under 5mins, we've seen where the time increases to 25mins

In a live match where the traffic is rapidly increasing we cannot wait till servers being provisioned because to hotstar customer we can't comment like there's no EC2 that's why match cannot be shown.

This is what they do, pre warm the infrastructure before the match.

They don't use the ASG which AWS provides instead they developed their own ASG.

What it does is , instead of scaling on default metric like CPU, N/W it instead scales on req rate and concurrency.

We get concurrency, based on total active users of the platform.

They've a ladder defined, 3M means these many servers for the application.

At 10M each application will have these many servers. Whichever is high req rate or concurrency, it will scale accordingly.

Because CPU shouldn't be a metric to scale up on because CPU unless it impacts the customer but if it's not increasing latency. CPU can be high 60 % or 70% but if its not creating the problem to the user, it shouldn't be a metric to scale up on.

Instead we've a bench mark on which each server or each container can serve with the greatest rpa and we take decisions based on that.

If 2M platform concurrency is doing 50kbps, if that's enough we can stay at that line.

If its now 75kbps, we can scale to meet the expectation.

Secondary Resource Group:

It's the problem related to single instance type. ASG doesn't have multiple instance types. We can spin up secondary ASG for that.

ASG gives the notification if its unable to increase the capability. That notification can be used to push to SNS to trigger a lambda function which automatically scales the secondary ASG.

This way if a single instance type isn't available we can spin up secondary instance type.

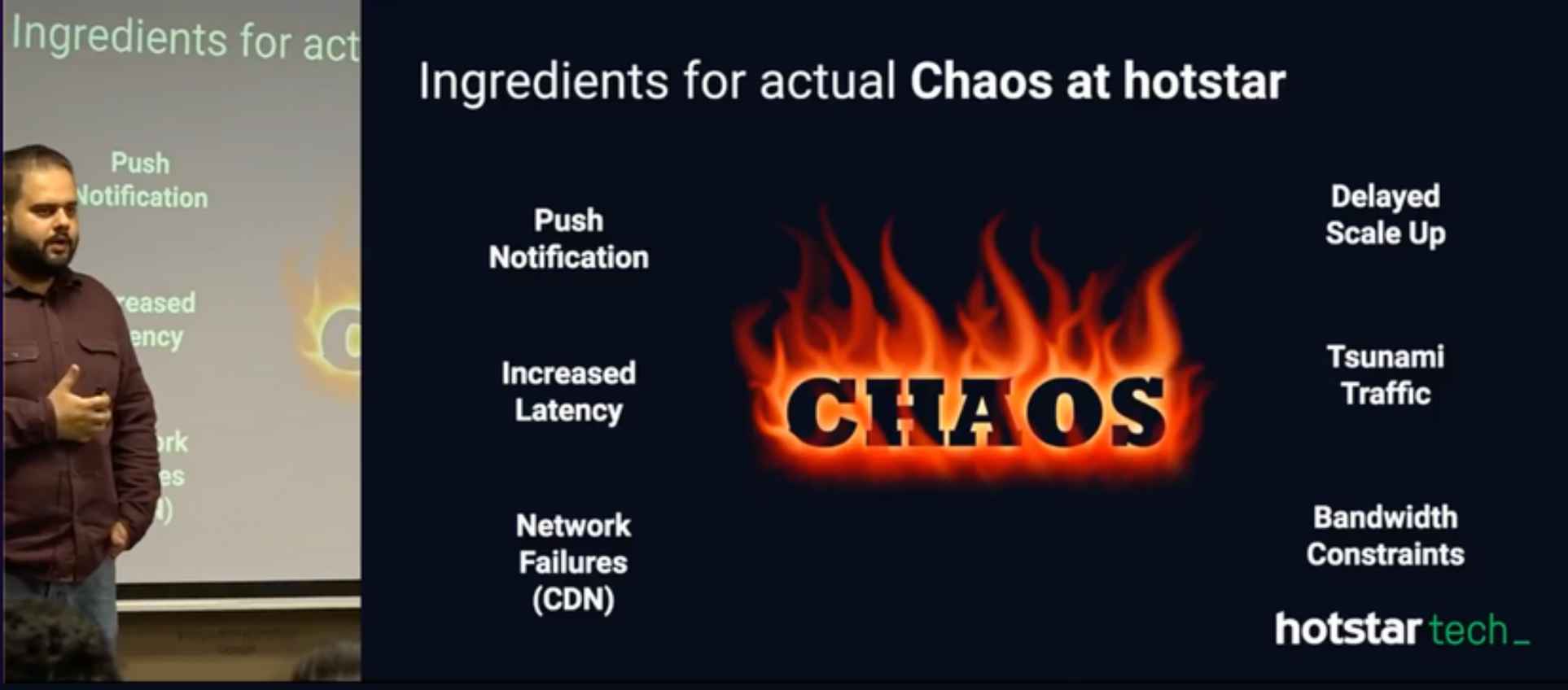
SPOT FLEET ADVANTAGES => Cost Savings, A Single Splot Fleet can allow 50 diff. instance types in a single container . This is before the EC2 came into play.

EC2 fleet is a mixture of all demand on spots.

By Splot Feet we re able to configure more than 1 instance type. We can have a compute and memory intensive family, (c4 4x, c4 8x, etc).

If one instance is not available in AZ, other instance from AZs can be used to fill in the capacity that may be required.

Chances of getting capacity errors will be less



Above pic contains the ingredients for CHAOS

CHAOS Engineering is the breaking point in the system or art of breaking things. So that we can know that the failure is going to happen and how can we overcome it from impacting user.

Marketing team can anytime send the push notifications, backend services need to cope with that spike.

Even in one application in entire user journey is impacted, it has a cascading effect on other services.

On the homepage there is specialization engine, recommendation engine which shows what content users had watched and what content users should watch.

If content platform api is increased later by 50ms.

There are other services that consume this api, will work slows and can result in slow app start time.

Hence, the single increase in latency of one single api has a cascading effect on all the other api's.

Network failures are another scary thing.

To operate at large scale, they depend on a lot of CDN providers.

What happens if the edge location goes down or overwhelmed, they've to reboot or shift traffic.

All the traffic serves to edge location closes to edge location or ISP. All those requests now come to origin.

Lets imagine if they operate at 10M, and the edge location closer to users home is down, requests will then directly come to the origin endpoint.

Even if it is half a million user or 5% of 10M, if the application is not scaled up to handle that origin request. Application may go crazy or DB can go idle. Because application isn't supposed to handle that sudden spike.

Delayed Scaling is the reason they went to the auto scaling because if a live matching is happening and they want to scale. They need a server.

If it's not available and scaling script takes time to scale up the infrastructure, users may be affected or they might get a bad experience.

Tsunamic Traffic is the first graph that we saw. It represents the sudden surge and the sudden dip.

We can still scale but think of the backend servers like cache, RDS. These are not scalable on the fly because there can be a downtime associated. It's not a thing we can do on the fly in between a live match.

Bandwidth constraint is a another problem, with more users coming in, we consume a lot of video bandwidth and the stats that we saw on the first slide more than 10TBps , that is almost 79% of India's capacity.

There is very a limited room to operate in, in terms of adding more users. Lets say the concurrency goes on from 25M to 30 or 40M users, is there enough internet bandwidth available to serve customers.

We can push out a video but if we're living in a area where latency slows down and ISP chokes or throttle.

May be 1k users watching hotstar on same ISP in the local area , this will be constraint for hotstar from bandwidth point of view.

The main aim of chaos engineering is not to just bring down a system.

Lets imagine if 1 AZ goes down and if there's some N/W issue b/w EC2 and DB, will the application still perform. Those kind of tests thy perform in a controlled environment

What are the worst case scanarios that can happen if anything goes wrong in the system.

Bottlenecks / chock points are related to anything which cannot be scaled in time specially data stores and backend systems which needs to be provisioned to a peak capacity because they are not that elastic in nature.

Breaking point is at what rps or tps, application can go down.

Then we take any action to avoid going or touching that number and how can be scale up some of the things by introducing a caching layer or do some api action that is not required to be done by the most later on by a backfilling process.

Death wave is again the sudden spike. Adding 1M users every minute it has a upper limit, it cannot infinetely scale up a system. For this they are prepared for a 50M concurrency

Above 50M is the territory for which hotstar is not prepared acc. to the video, the threshold can go upto 35 or 45 M

Failures can happen at any level : n/w, servers or applications. By doing this chaos engineering exercise what hidden patterns are in each of the areas. Then they try to fix that.

The other good thing about rated capacity is once we know the user journey, that once we open the hotstar app, that these are x number of api calls that users can make.

If we don't know the user journey, we can't scale the infrastructure resilient.

If application A can handle only 10M load we can turn down the application before it reaches 10M to avoid a outrage stuff like that. Using CDN or elastic cache outrage can be managed.

Panic mode has a basic principal that key services should always be up.

Users shouldn't be notified if something is down such as the RDS.

Panic is where we turn off non critical services to make a room for critical one.

When 25M people watching cricket thats about 99% of traffic active on platform that time. So for 0.1% it doesn't matter if they need personalized content or non essential things can be turned off thereby making room for critical api services via key health check metrics and concurrency pattern.

P0 services should always be up , for them p0 is video, advertisement, subscription and payment systems.

Non essential sections like recommendations or personalization can be turned off.

We can this way make bandwidth available for p0 services by reducing traffic at other services.

Graceful degradations is for applications where we can handle only 10M traffic and it doesn't make sense to scale it up more by adding hardware capacity unless it adds a business value.

In that case we can cut off a system and return 200 OK value so that client doesn't create a error message and shows user a bad message.

Clients are also smart enough to know that the application is in panic mode. It will give a custom error code.

Let's say if DB has a issue and users are not able to complete the payment transaction which is a known issue and panic issue can be put at server end and client if it retries, it knows that there's a issue at the backend.

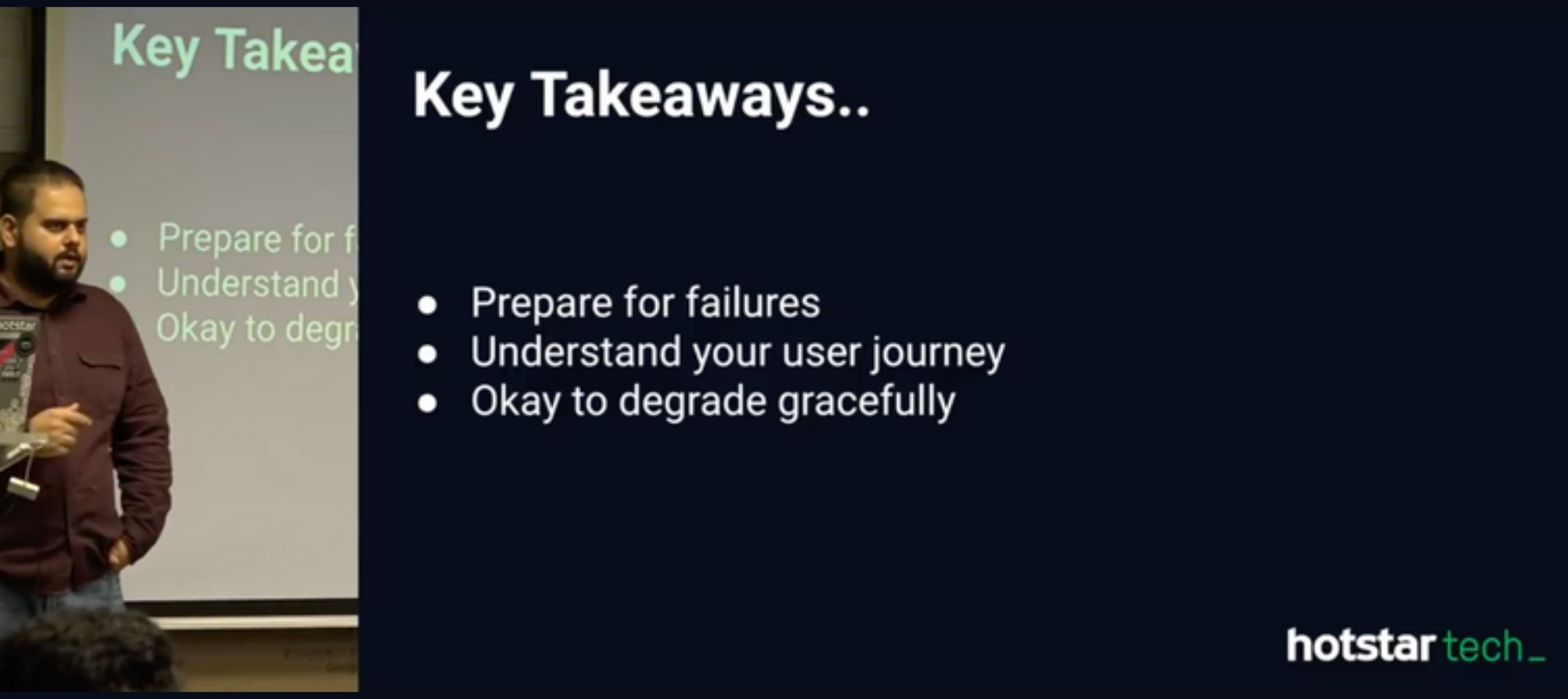
If in the login system if dynamoDB or ELB system has a issue, we can put entire login system into panic. It allows user to watch the video w/o even asking them to login.

They can bypass users from login, till then they can fix the issue. They can disable the panic mode so that normal system flow will continue.

After 10M there are decisions taken to understand whether service or application is necessary for business. And to check whether it has rated capacity. If there are 80 or 90% of rated capacity, they'll put the system into panic and degrade them gracefully. It'll not have a cascading effect on other

services.

Failures are bound to happen and they are not under control. But it's the developer job to create load testing pattern and handle them.



When something is broken, system can be degraded gracefully so that user can be unaware of that.

If Dhoni comes to bat, they had to think that way during the match and be ready to scale up.

It took almost 2 years for them to build this kind of scalable system and for many of the unique problems they didn't had any references, they had to do RCA & POC as no one has done that with scale even with CDN partners.

For Game day they use 3000 c5 9x machines and they are scripted to generat such a load that these concurrent users will log into the system

The logs in the access pattern gives like 5M concurrency, video applications will have about 2M and other have lesser concurrency.

If everyone is watching match, personalization will not have that much risks.

At each ladder, every application has a rpm or rpl, which using it operates.

If we find the ratio we can mimic the entire graph that we saw in the first vide

Their CHAOS engineering is fully scripted. Mostly its python based. They are open source tools available for this. What they tried to achieve via CHAOS is:

1. Error that has happened in past, what if it occurs again. They simulate to check N/W failures and DB logs.
2. Sometimes they just go and randomnly does things thats not even thought of.
3. If a system is talking to another system via VPC peering , we can delete that N/W connectivity and we can go and check route tables to just verify that the application can handle whether the load or performance if N/W connectors doesn't work.
4. Its a python bot script for automation

PANIC mode is a feature toggle they've implemented at client level and at server side it will return OK response instead of 4xx or 5xx.