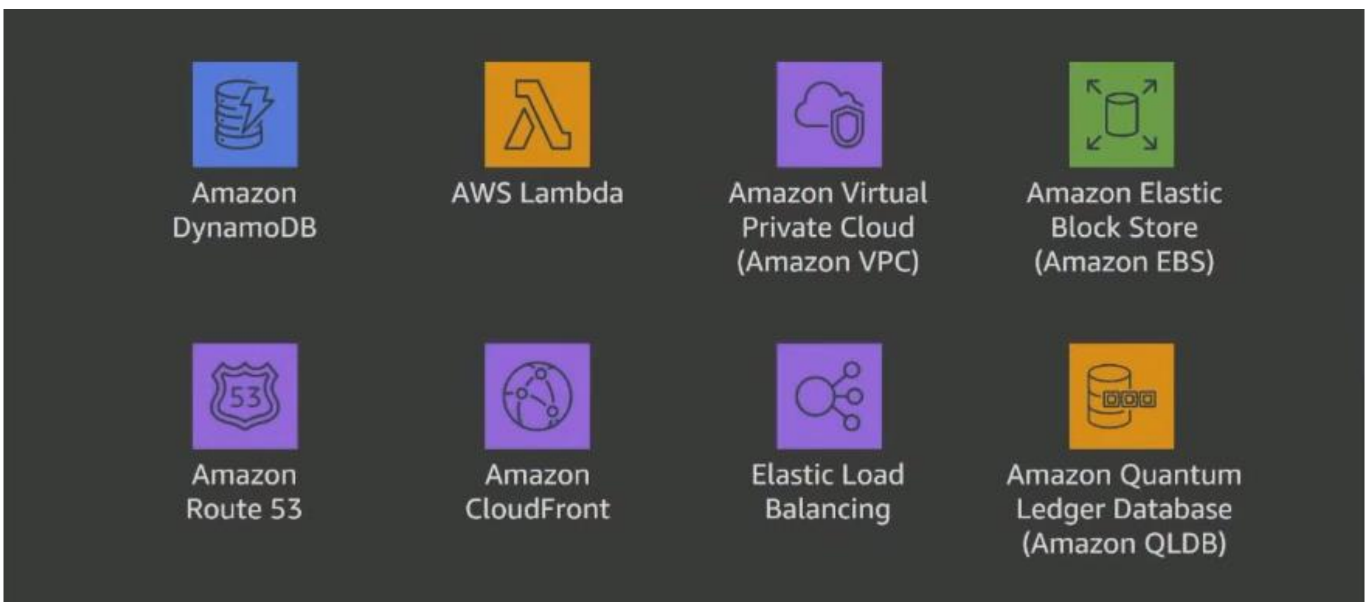
The Amazon Builders' Library is a collection of living articles that are written by Amazon senior technical leaders and engineers and that describe how Amazon develops, architects, releases, and operates technology. In this session, five principal engineers share hard-learned lessons from their experiences building reliable services at Amazon. David Yanacek, Andrew Certain, Becky Weiss, Colm MacCarthaigh, and Marc Brooker each share a personal story highlighting a current best practice. The engineers discuss how Amazon uses timeouts, how we think about back-offs and retries, our approach to taking dependencies, how we measure performance, and how we use shuffle sharding.



Lightning talks

Monitoring with percentiles

Static stability

Shuffle sharding

Retries, backoff, and jitter

Meta lesson: Learn from each other

Lightning talks

A. Andrew

B. Becky

C. Colm

D. David

E. Marc

Monitoring with percentiles

http://blog.tacertain.com/p-four-nines/

Tim Bray
@timbray

Was in a conversation about some service refactoring and the guy said "mean latency was a bit worse, but our P-four-nines was rock-solid and only twice the mean. Customer loved it." #CloudThinking

132 3:09 PM - May 24, 2019

See Tim Bray's other Tweets

"P-four-nines"

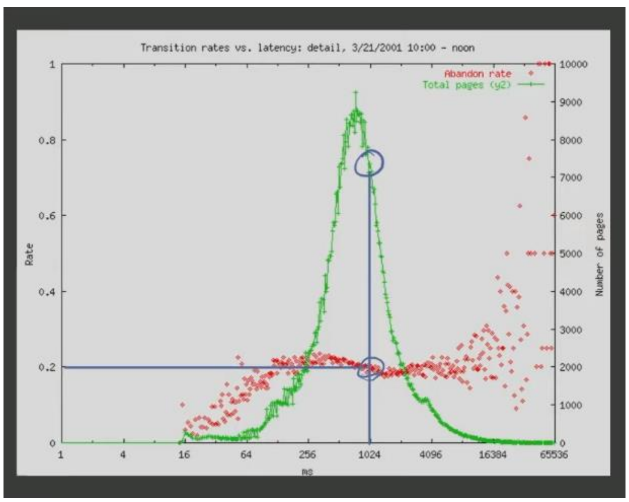
=

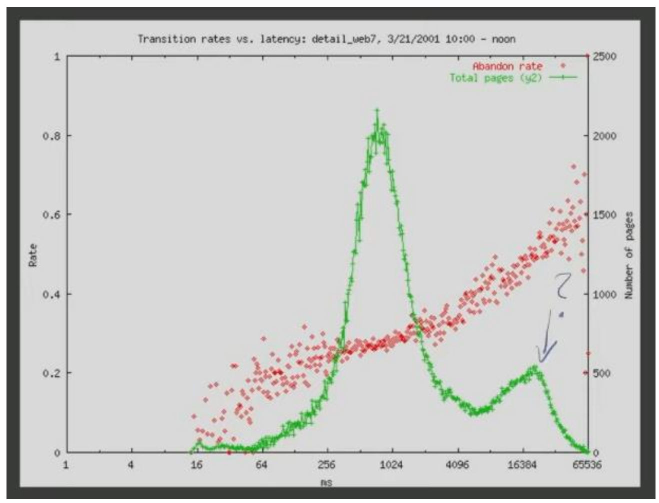
99.99th percentile

=

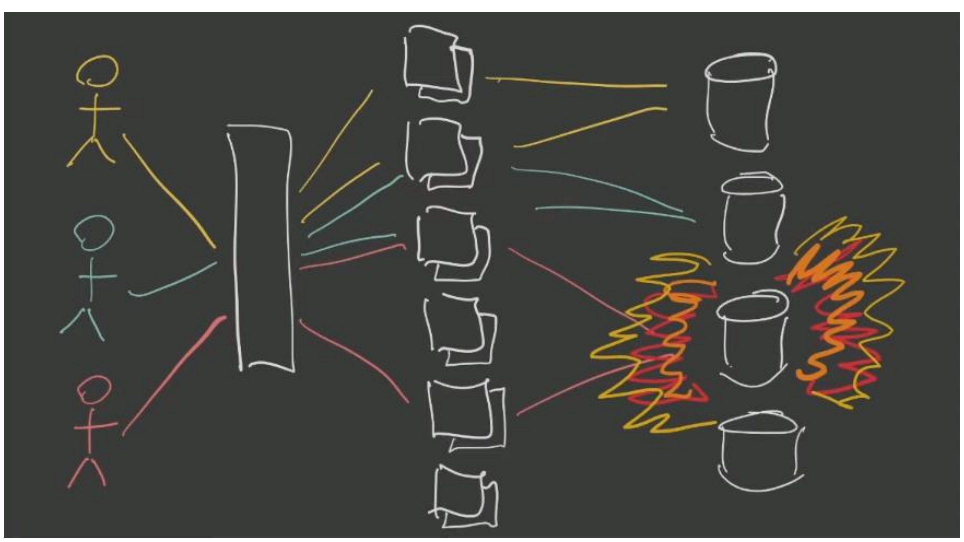
Value that is >= 99.99% of the data points

This is the 2001 version of the Amazon website. We switched from looking at the average values to using percentiles

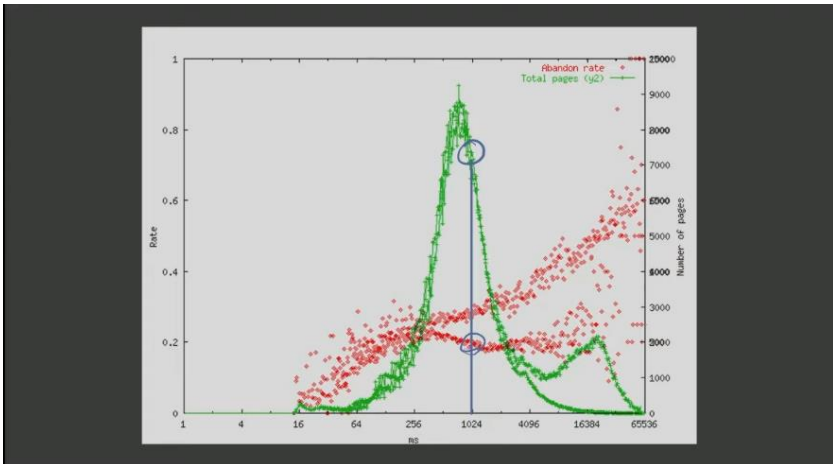




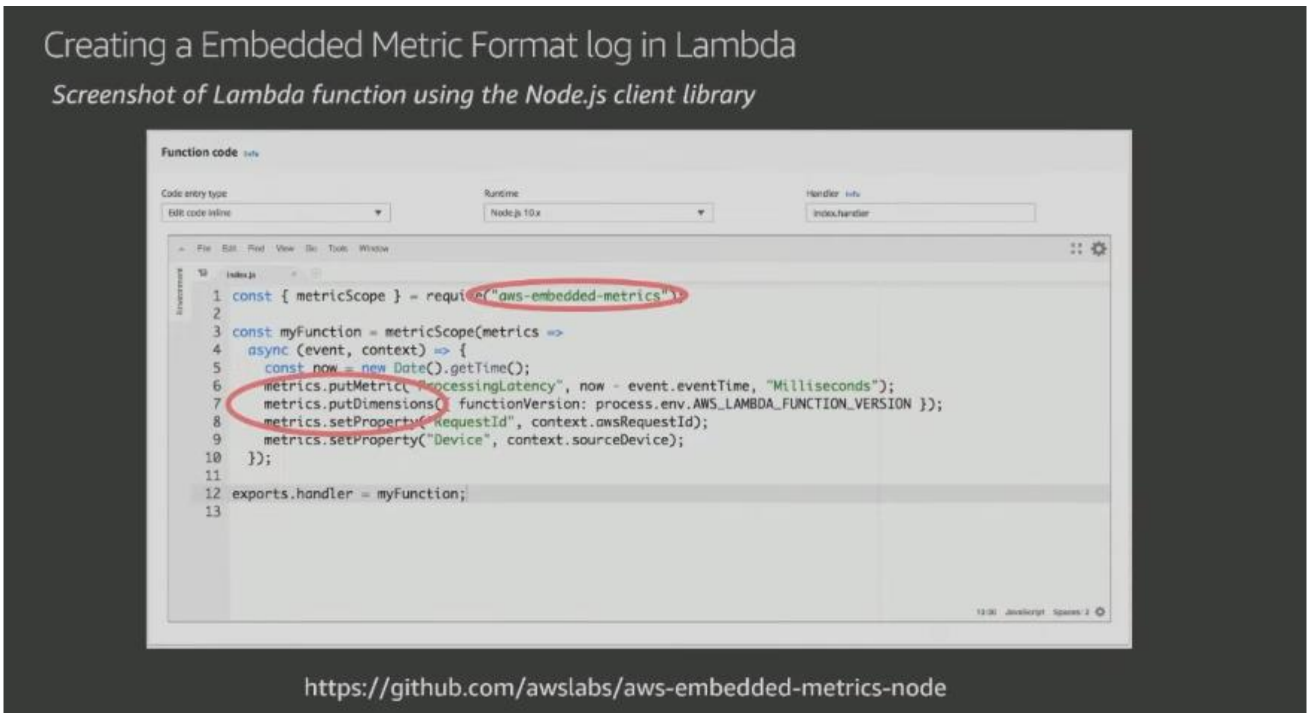
About 7000 pages had 1 second latency. Less latency means that the abandon rate increases



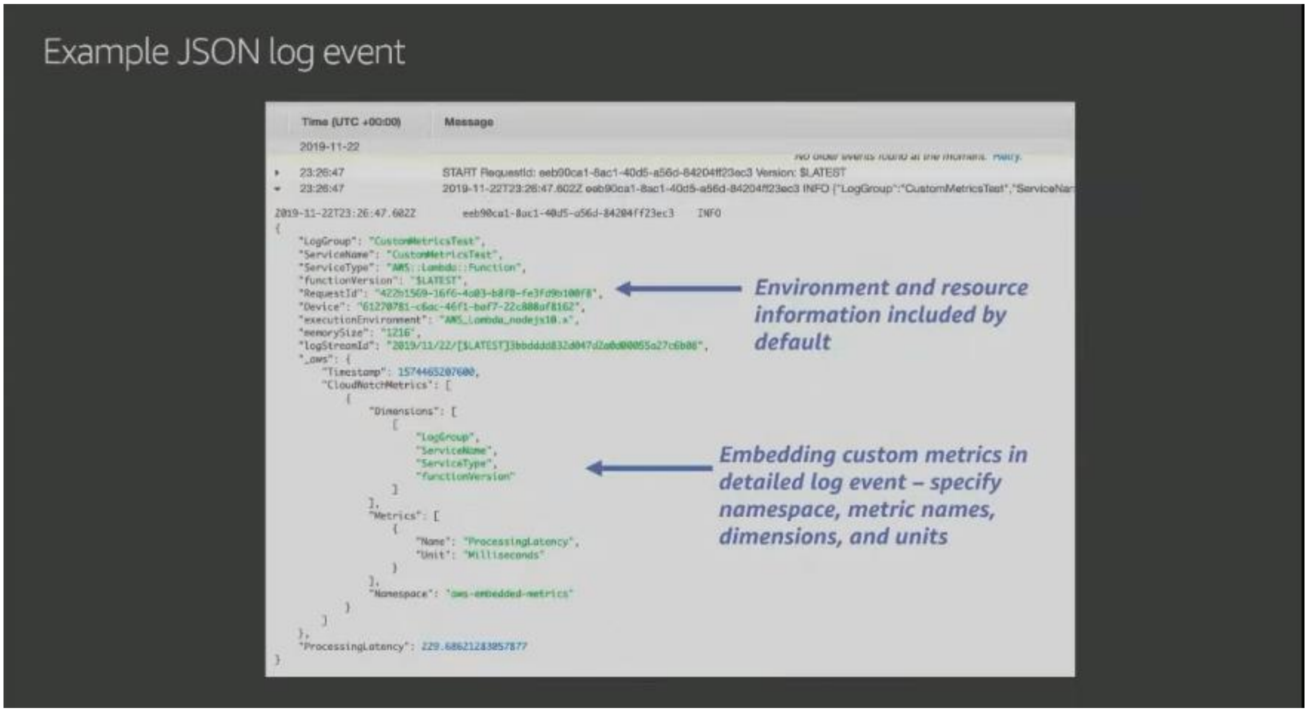
This is a rendering of the website in 2001, LBs, web servers and the databases. One DB was having troubles and adding latencies to the page load times randomly for the page views.

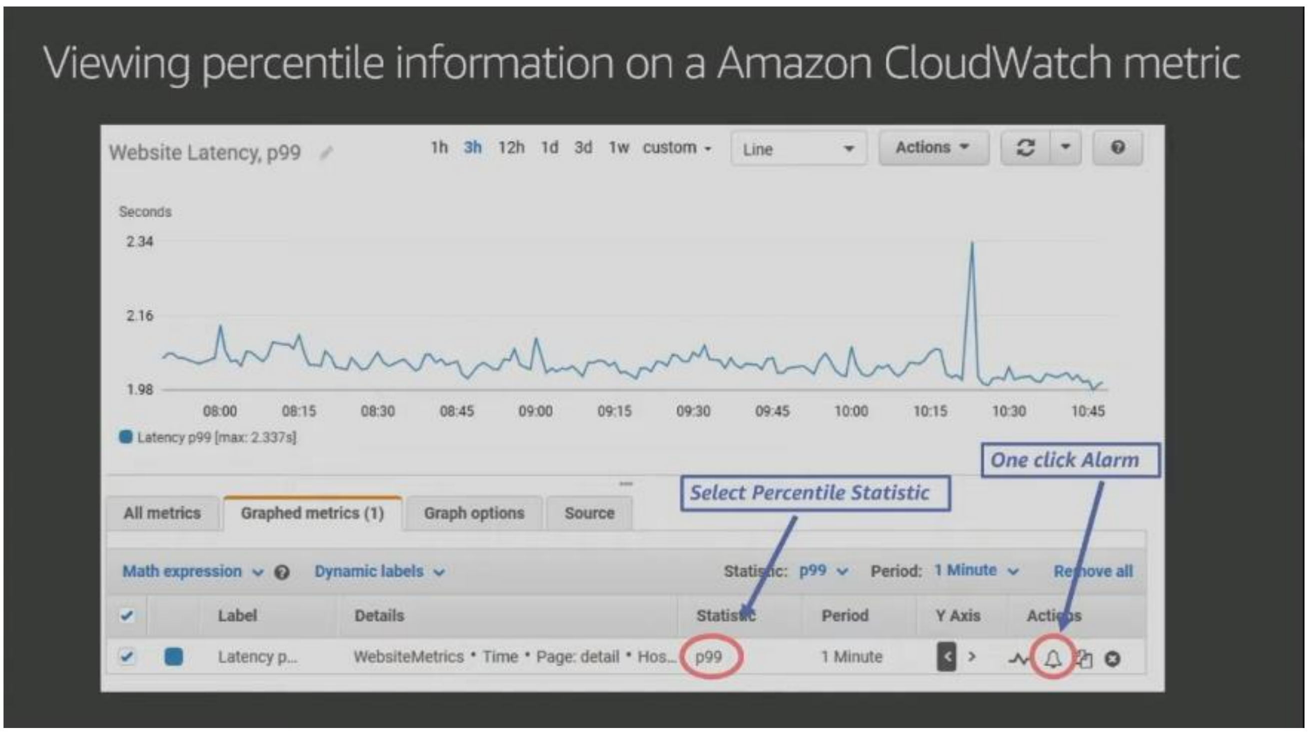


The bad DB was causing the high abandon rates when the customers were viewing pages that interact with it. We determined that the user threshold is about 6 seconds for the page to load, we need to switch to percentiles and use the 99.99 percentiles.

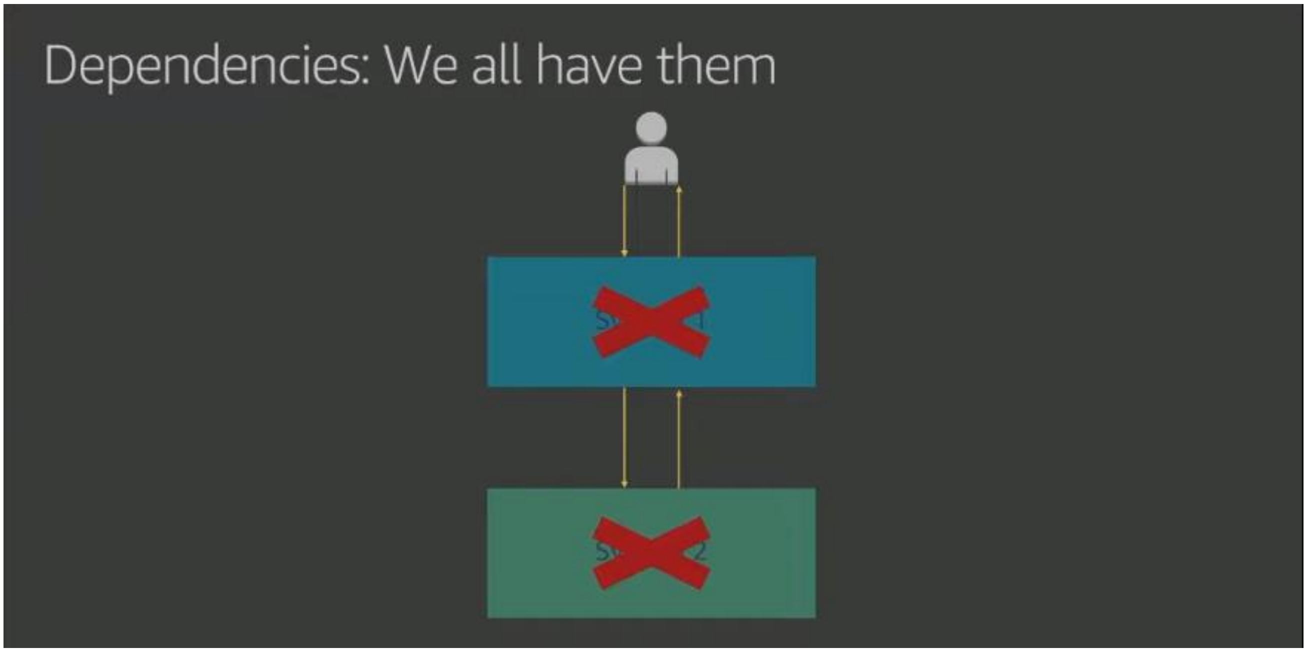


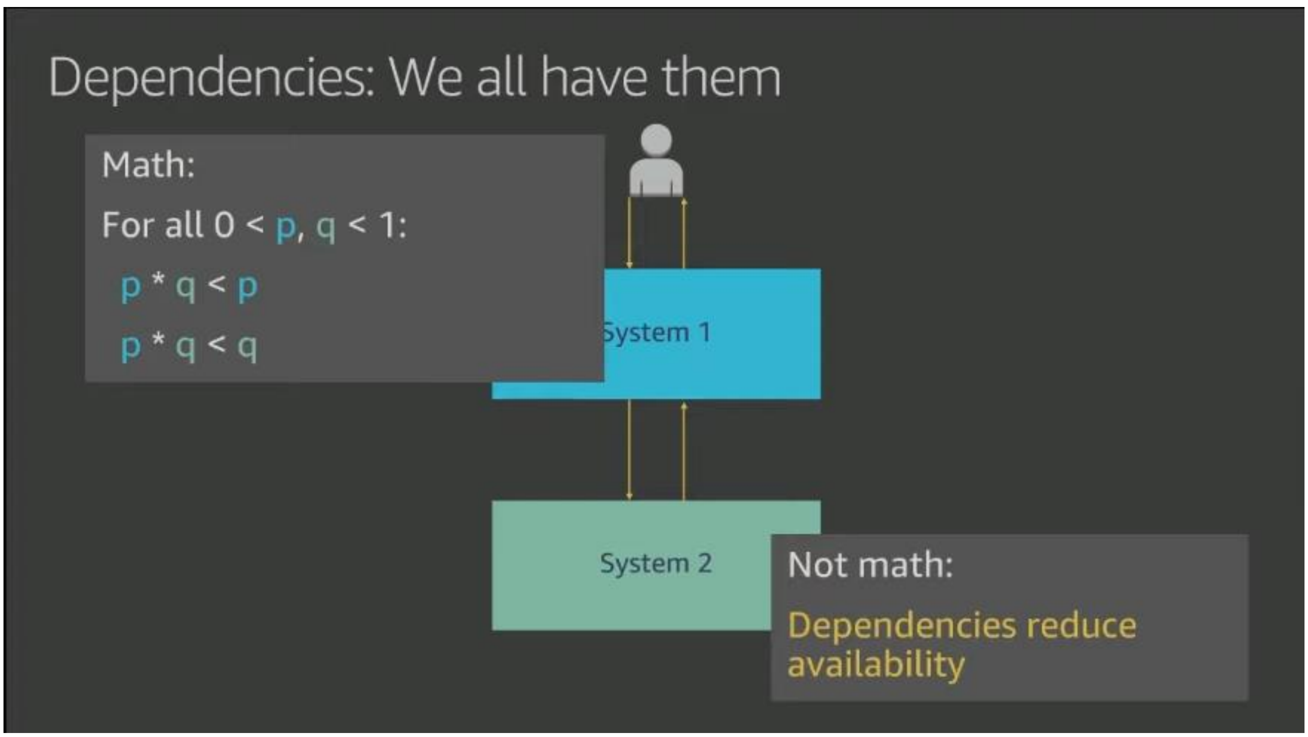
You can get percentiles data using a Lambda out of CloudWatch

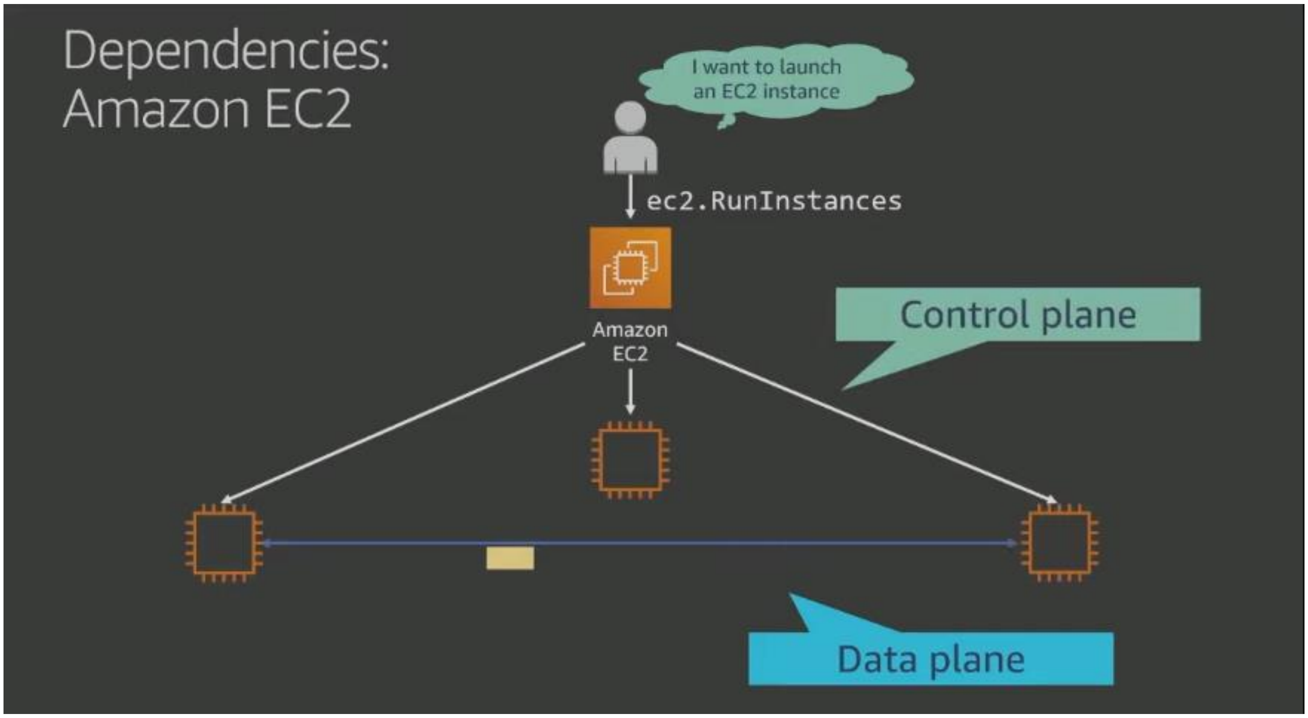

Example JSON log event
Time (UTC +00:00)
Message
2019-11-22
23:26:47
START RequestId: eeb90ca1-8ac1-40d5-a56d-84204ff23ec3 Version: $LATEST
23:26:47
2019-11-22T23:26:47.602Z eeb90ca1-8ac1-40d5-a56d-84204ff23ec3 INFO {"LogGroup":"CustomMetricsTest","ServiceNa
2019-11-22T23:26:47.602Z eeb90ca1-8ac1-40d5-a56d-84204ff23ec3 INFO
{
"LogGroup": "CustomMetricsTest",
"ServiceName": "CustomMetricsTest",
"ServiceType": "AWS::Lambda::Function",
"functionVersion": "$LATEST",
"RequestId": "422b1569-16f6-4a03-b8f0-fe3fd9b100f8",
"Device": "61270781-c6ac-46f1-baf7-22c808af8162",
"executionEnvironment": "AWS_Lambda_nodejs10.x",
"memorySize": "1216",
"logStreamId": "2019/11/22/[$LATEST]3bbdddd832d047d2a0d00055a27c6b08",
"_aws": {
"Timestamp": 1574465207600,
"CloudWatchMetrics": [
{
"Dimensions": [
[
"LogGroup",
"ServiceName",
"ServiceType",
"functionVersion"
]
],
"Metrics": [
{
"Name": "ProcessingLatency",
"Unit": "Milliseconds"
}
],
"Namespace": "aws-embedded-metrics"
}
]
},
"ProcessingLatency": 229.68621283057877
}
Environment and resource information included by default
Embedding custom metrics in detailed log event – specify namespace, metric names, dimensions, and units


Viewing percentile information on a Amazon CloudWatch metric
Website Latency, p99
1h 3h 12h 1d 3d 1w custom
Line
Actions
Seconds
2.34
2.16
1.98
08:00 08:15 08:30 08:45 09:00 09:15 09:30 09:45 10:00 10:15 10:30 10:45
Latency p99 [max: 2.337s]
One click Alarm
Select Percentile Statistic
All metrics
Graphed metrics (1)
Graph options
Source
Math expression
Dynamic labels
Statistic: p99
Period: 1 Minute
Remove all
Label
Details
Statistic
Period
Y Axis
Actions
Latency p...
WebsiteMetrics • Time • Page: detail • Hos...
p99
1 Minute

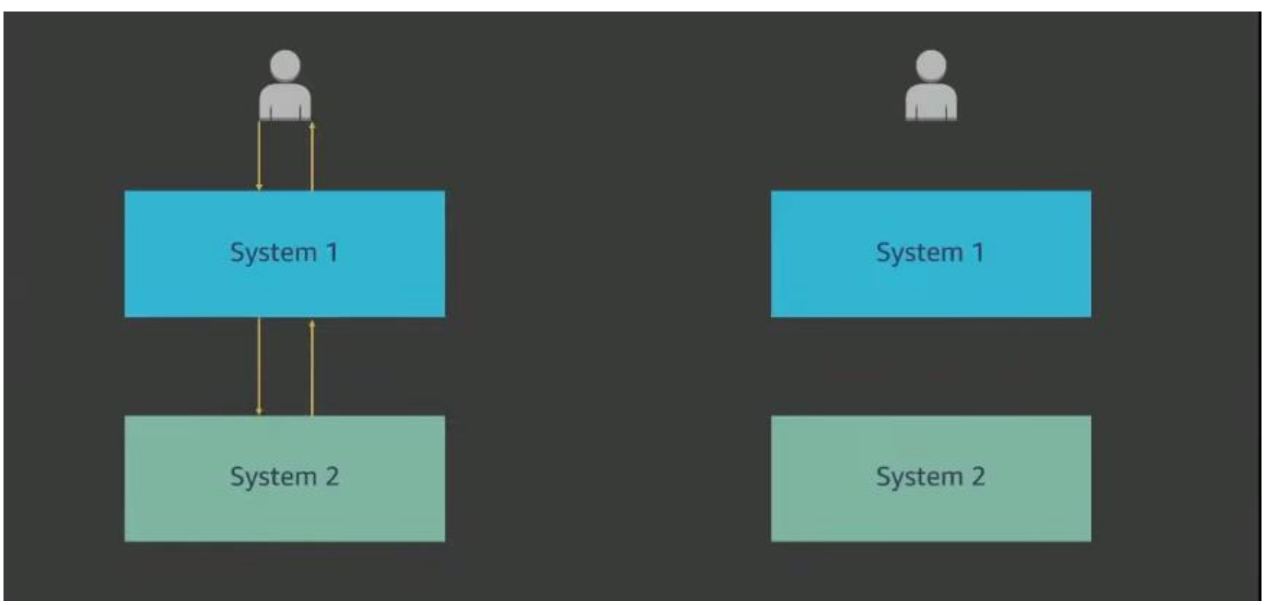

Static stability in AWS:
Resilience in the face of impaired dependencies


Dependencies: We all have them

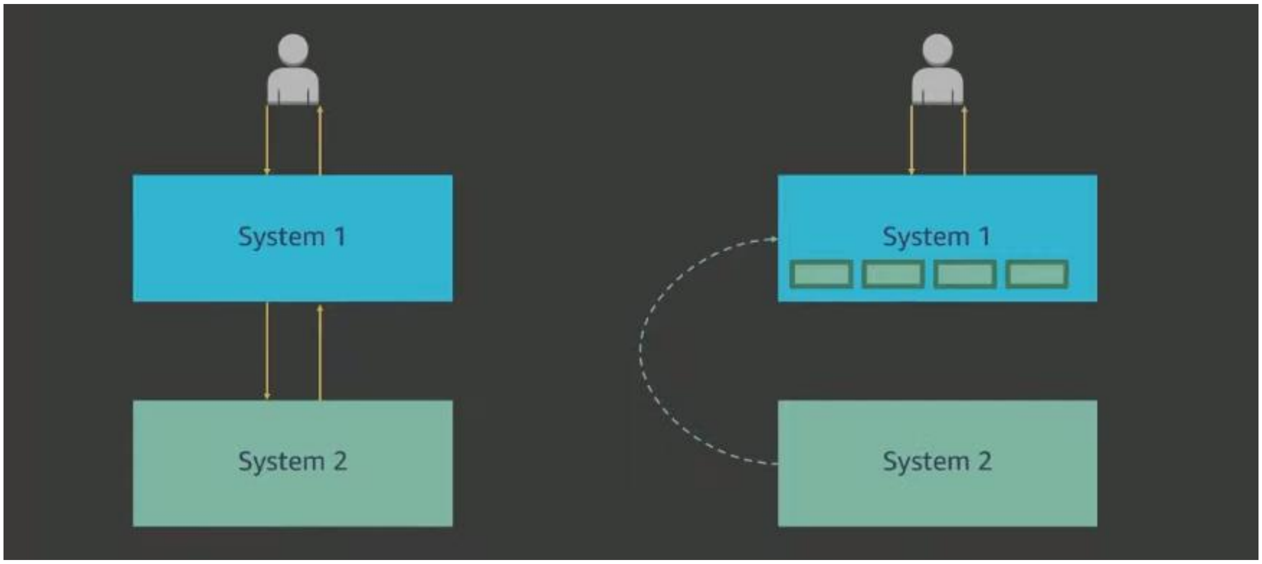

Dependencies: We all have them
Math:
For all 0 < p, q < 1:
p * q < p
p * q < q
System 1
System 2
Not math:
Dependencies reduce availability



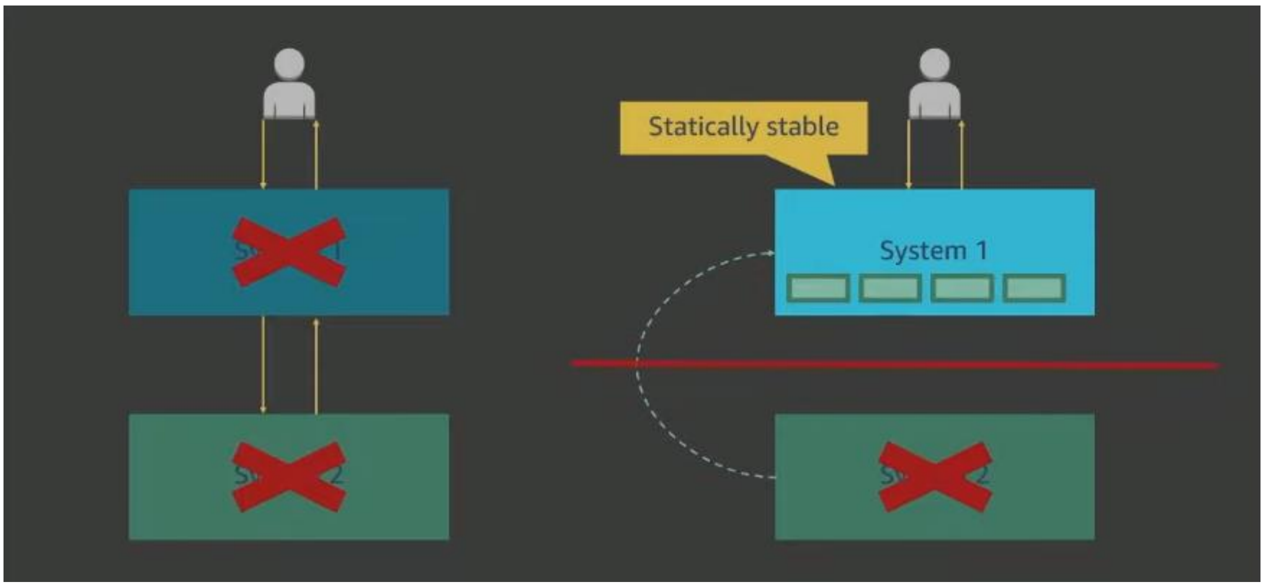
You might have 2 EC2 VMs running your apps with dependencies among them. We need to set up your Control plane for all these to happen for your Data plane to work.



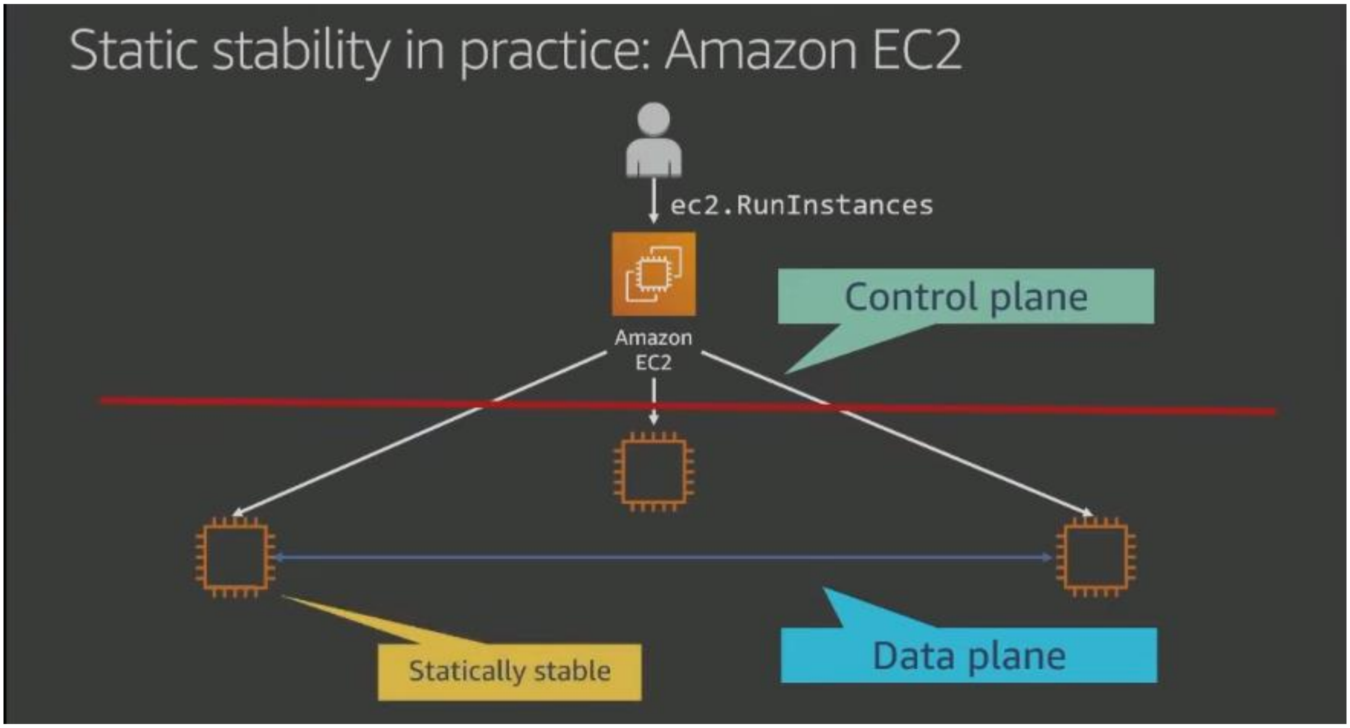
There is a pattern in AWS for being able to use the data from another system but also being resilient in the case of data unavailability of system 2.



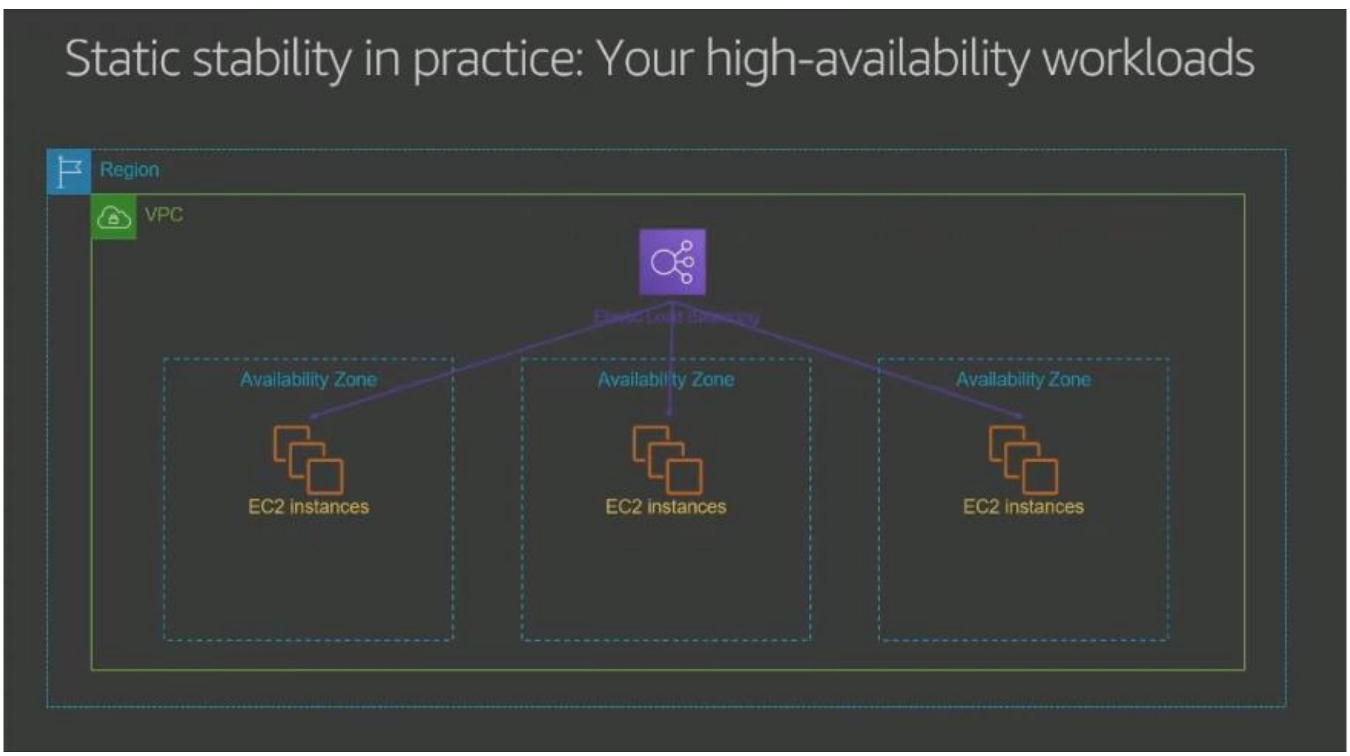
You can get the data into system 1 asynchronously so that it becomes available locally within system 1.

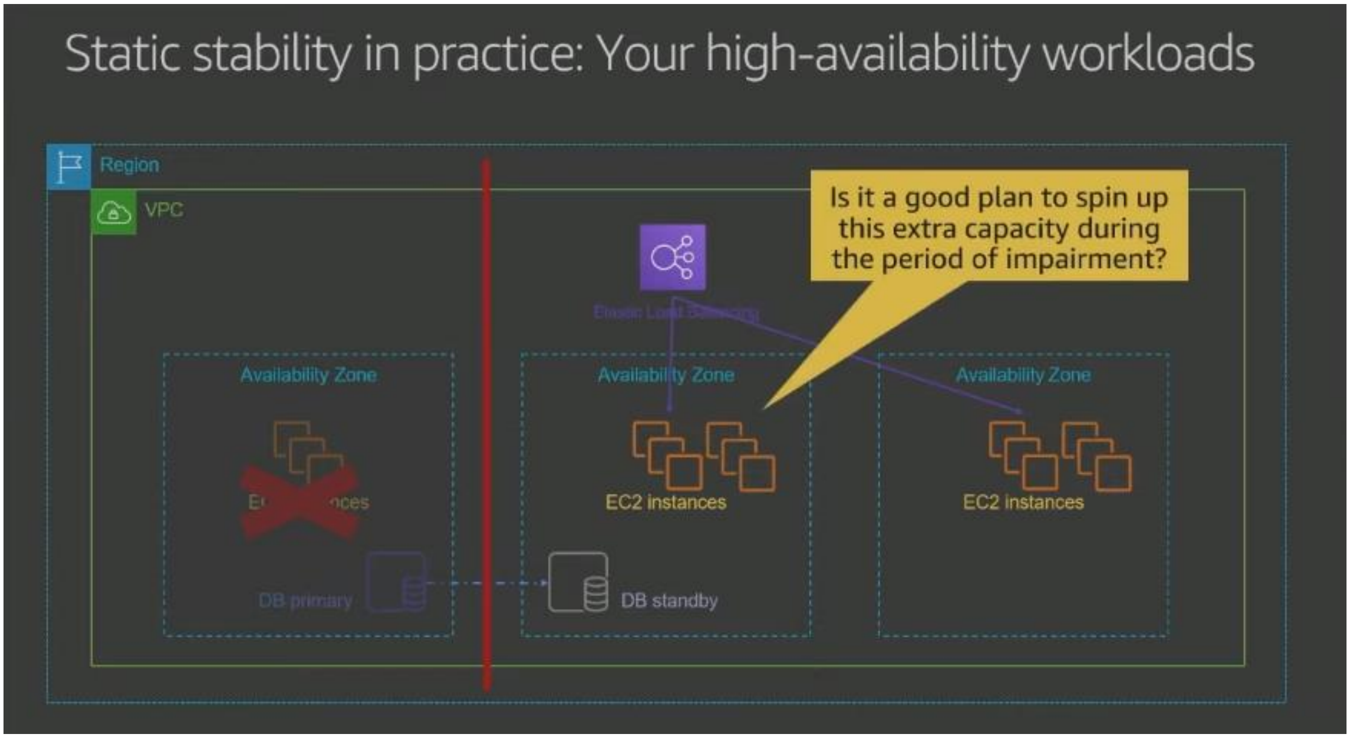


Now system 1 is still working albeit not updated



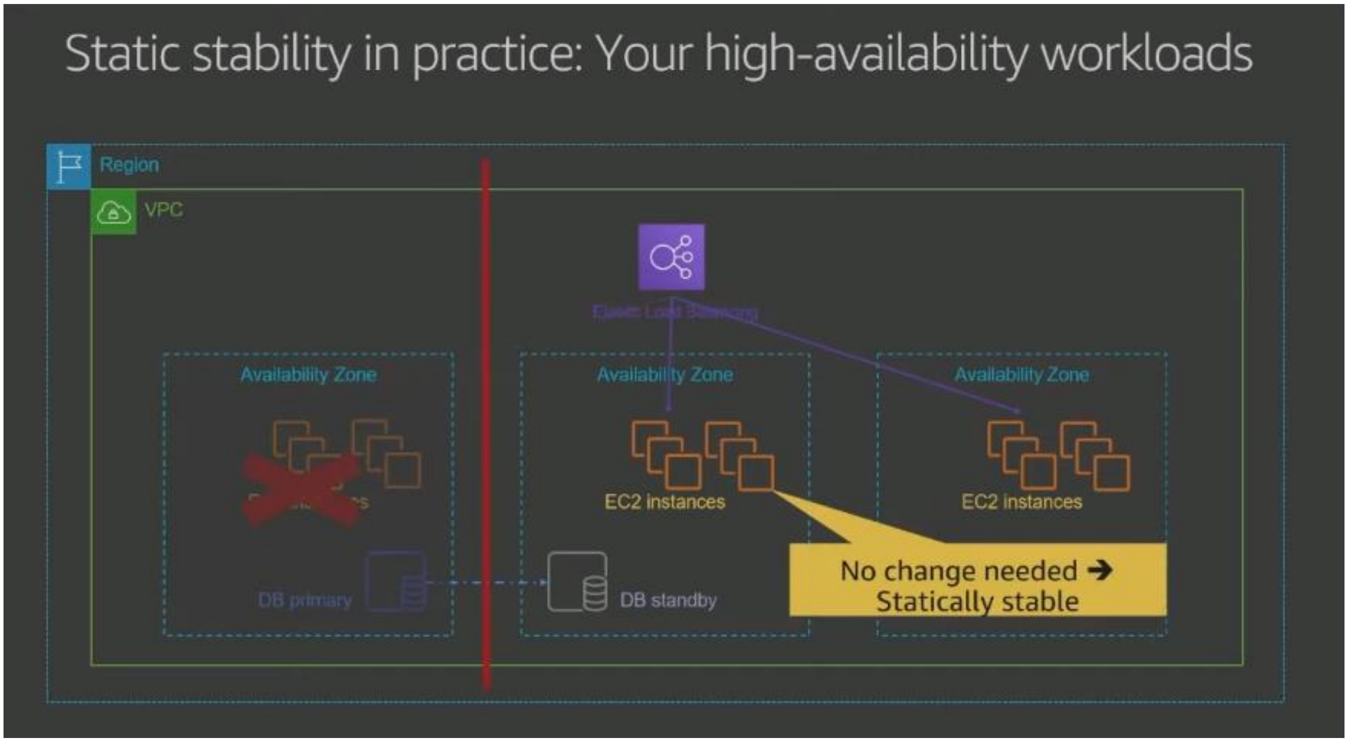
The Data plane is built to be statically stable, so that the Data plane doesn't even know about the existence of the Control plane.

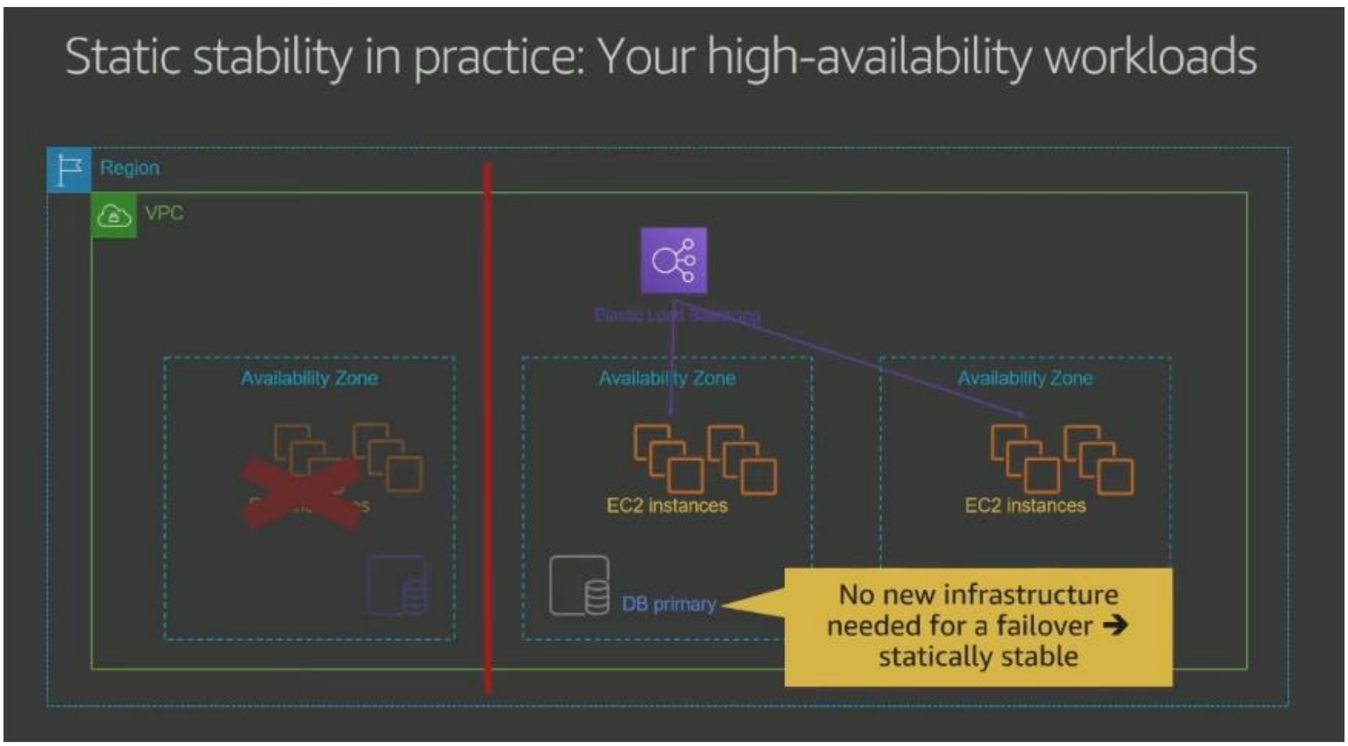


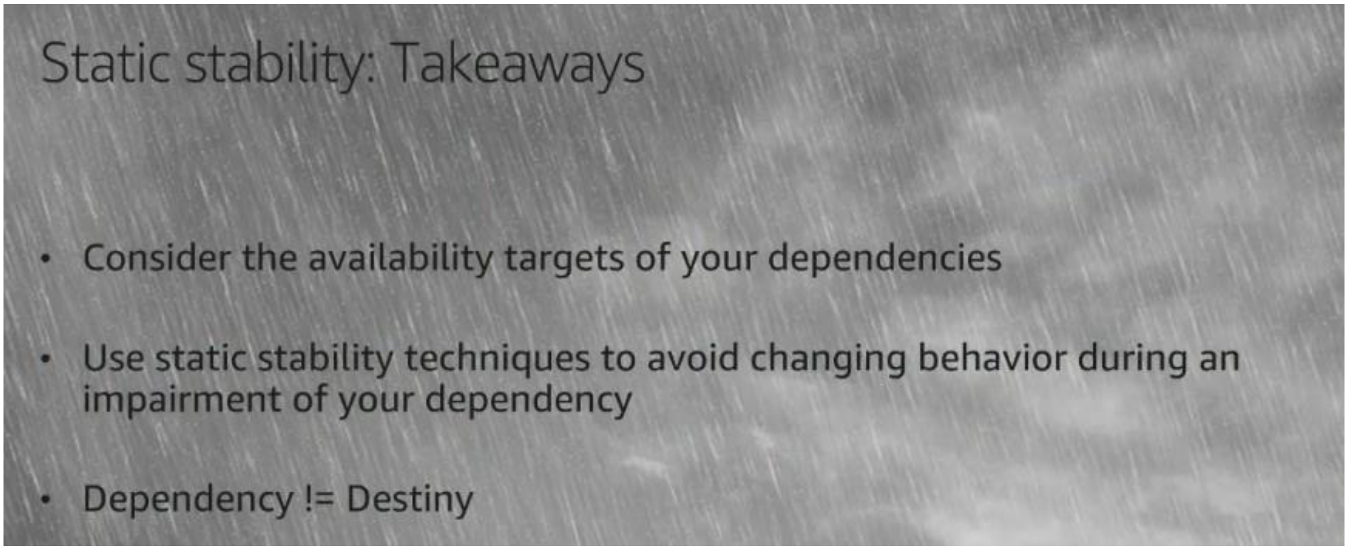


This is a rare event but possible.

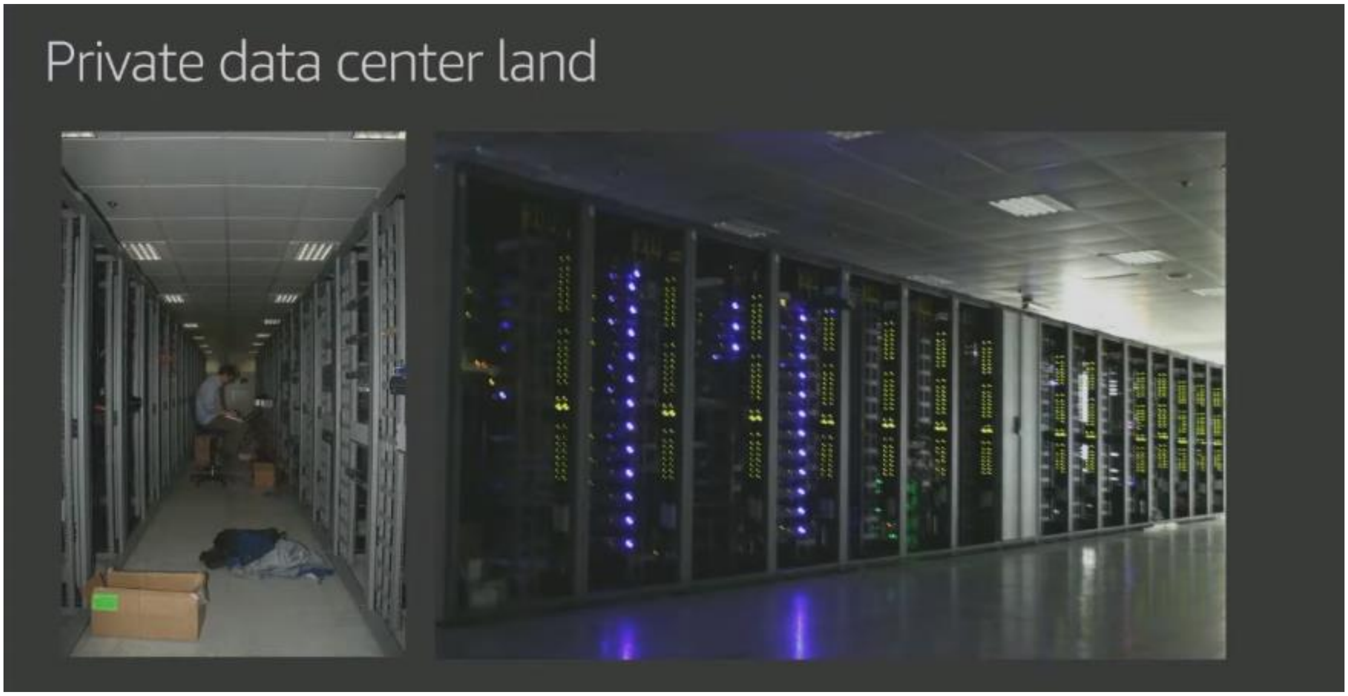
We recommend that you not respond to an incident while it is raining but you should already be ready and planned for failure before it happens.

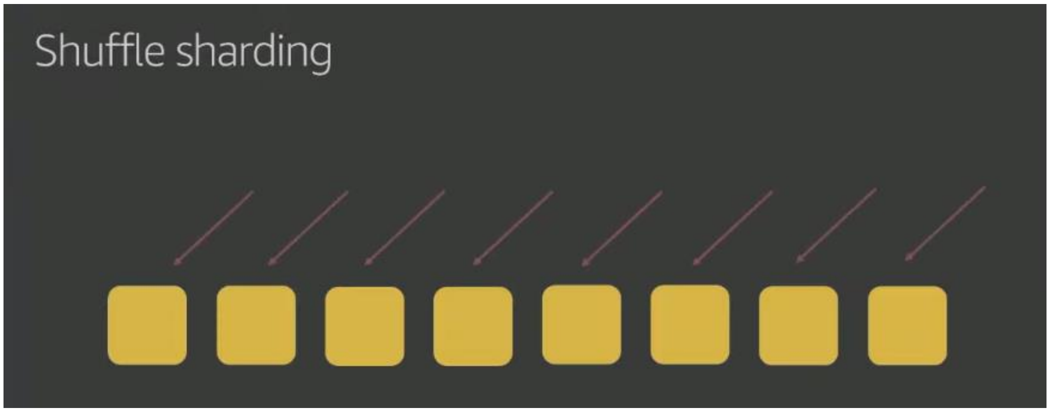

Static stability in practice: Your high-availability workloads
Region
VPC
Availability Zone
Availability Zone
Availability Zone
EC2 instances
EC2 instances
DB primary
DB standby
No change needed ➔
Statically stable


Static stability in practice: Your high-availability workloads
Region
VPC
Availability Zone
Availability Zone
Availability Zone
EC2 instances
EC2 instances
DB primary
No new infrastructure
needed for a failover ➔
statically stable

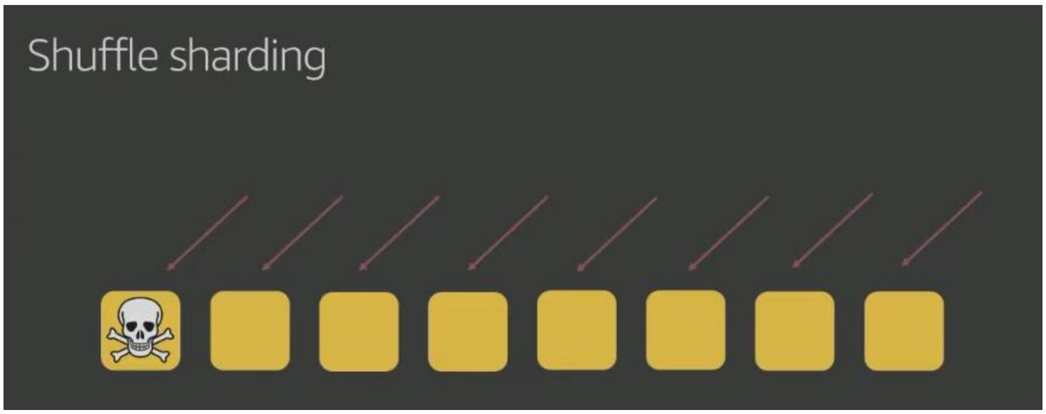


This is a technique we use to isolate customer workloads from each other, this is one of the core competency of a data center management system that serves multi-tenants.

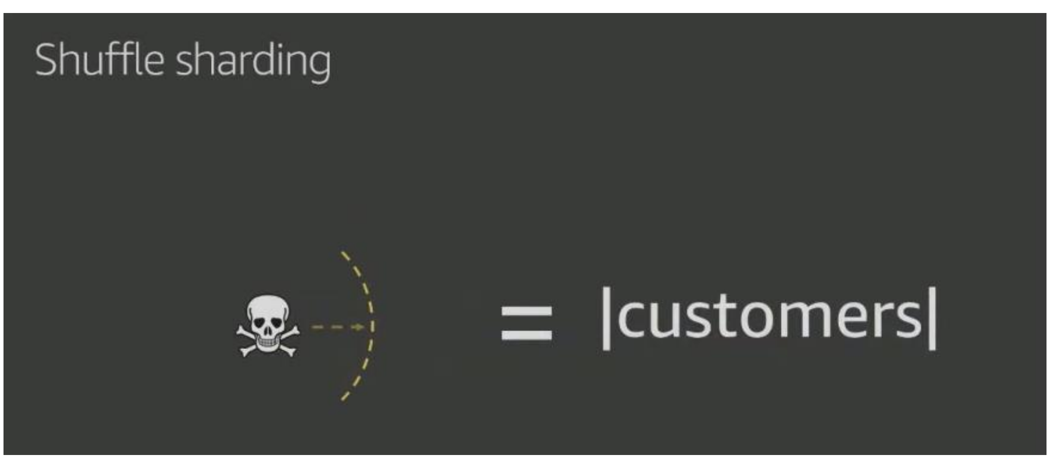


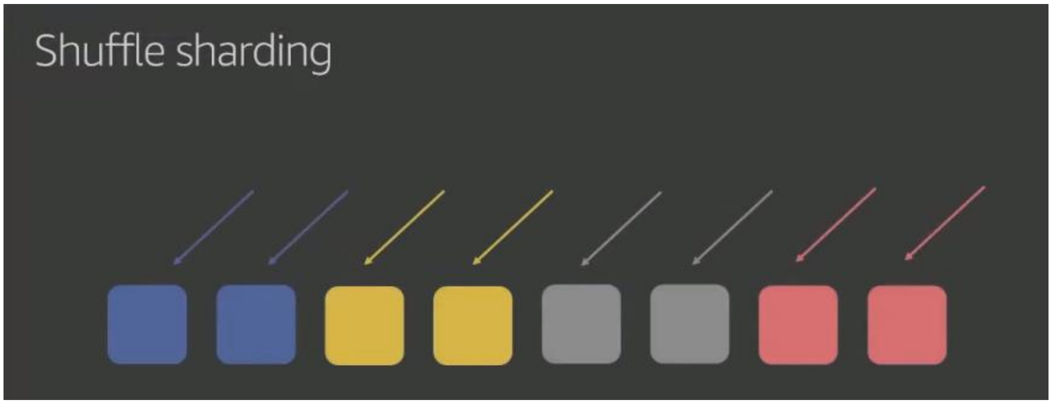


Assume we have some web servers doing work

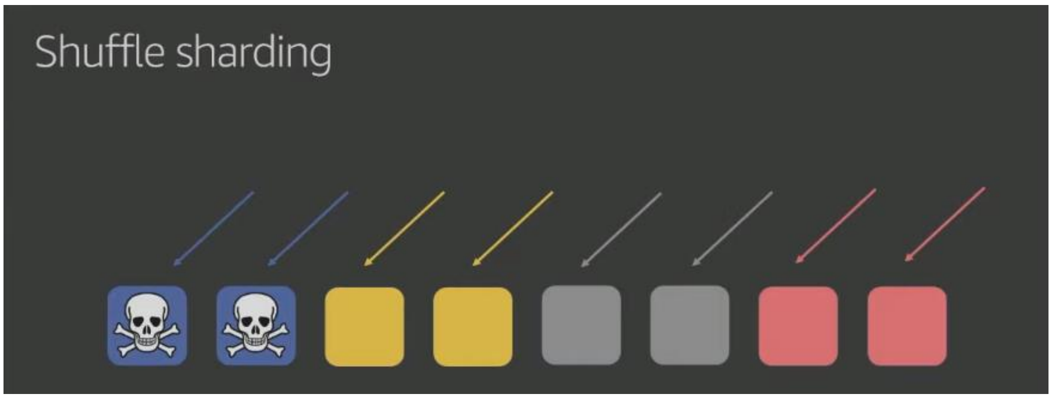


Assume there is a bad all within a server that shuts it down, the workload gets transferred to another server that gets shut down too

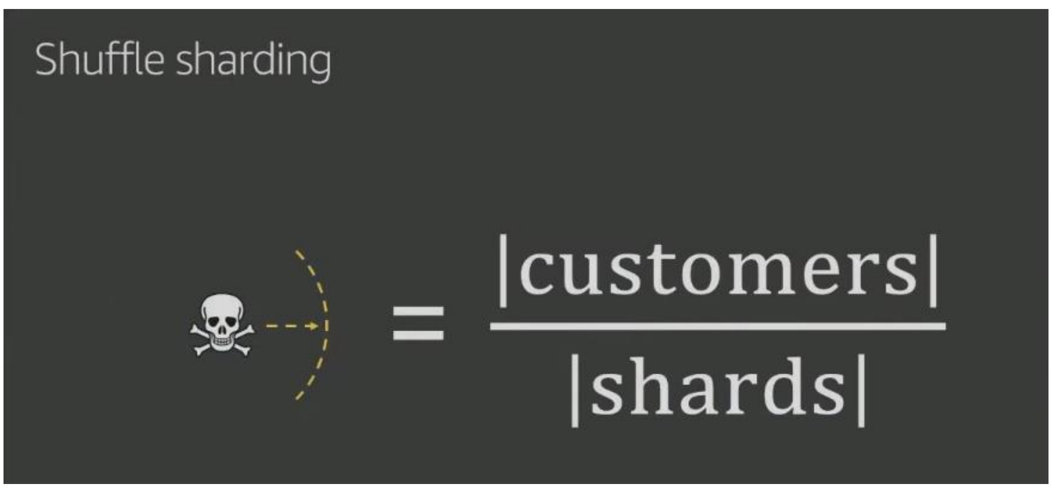




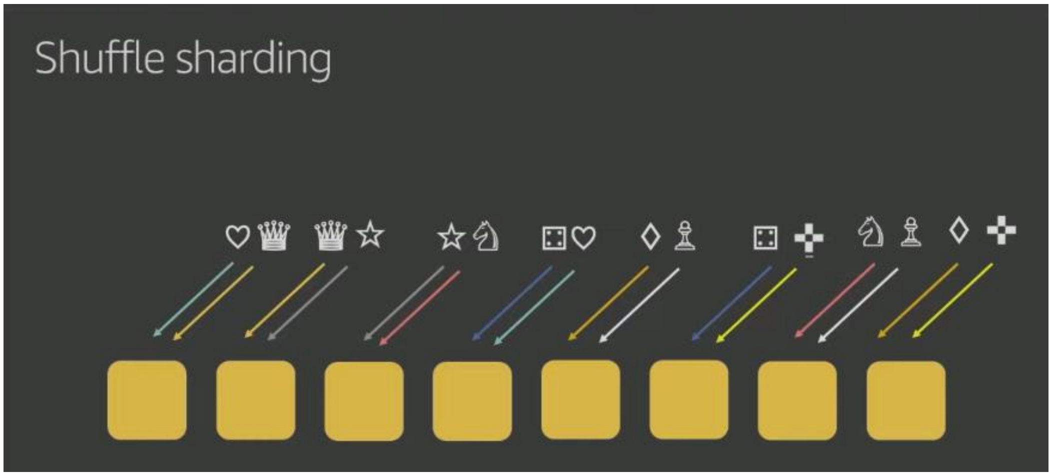
You can do ordinary sharding by dividing into fleets and assigning customer workloads to each shards

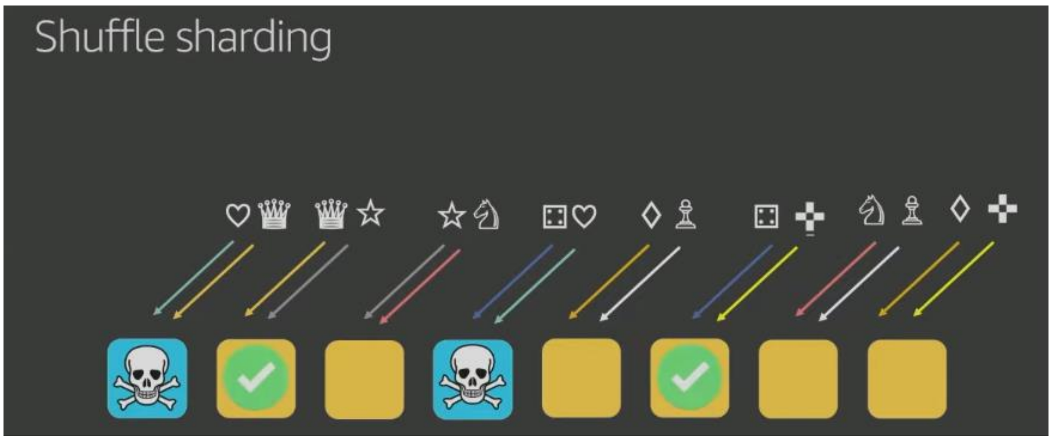


Now the other customers are isolated

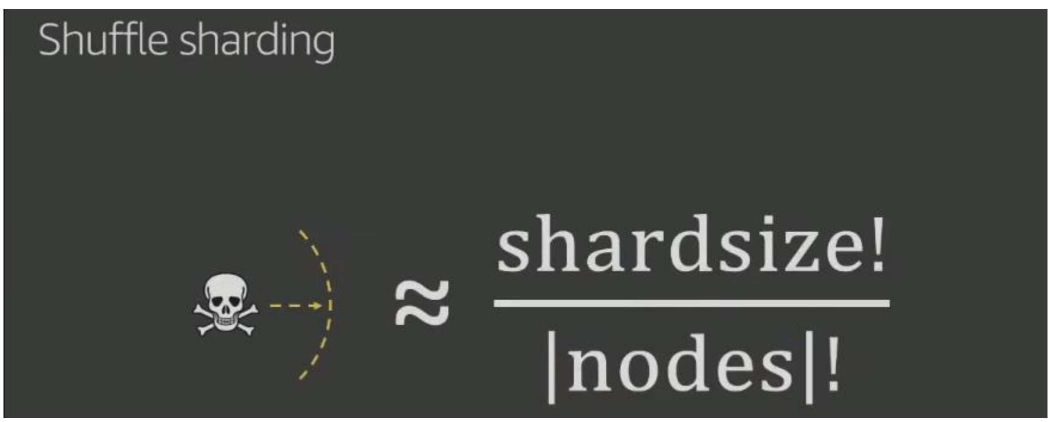


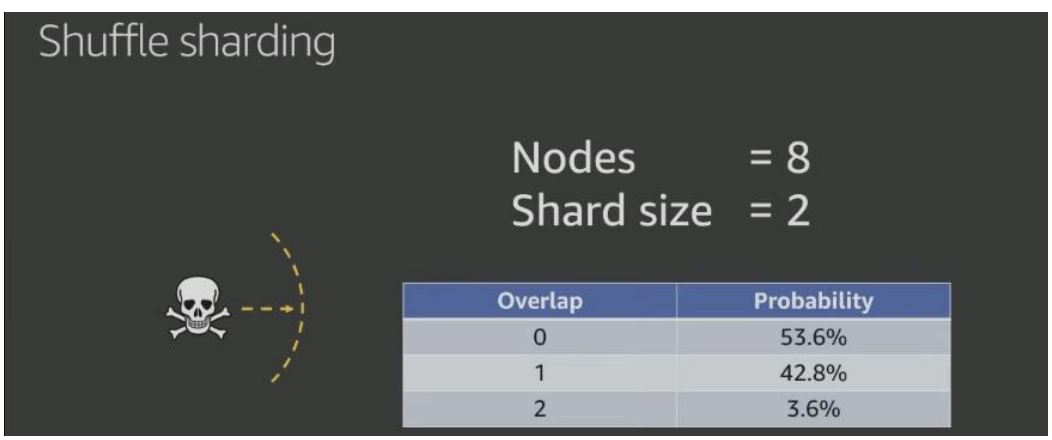
We can do more than this with the same number of resources using shuffle sharding, we use maths to assign customers smartly to a pair of server nodes





All you need is that the apps have some form of retry logic in them to allow the affected neighbors to get served elsewhere





Shuffle sharding

$$\approx \frac{\text{shardsize}!}{|\text{nodes}|!}$$

Shuffle sharding

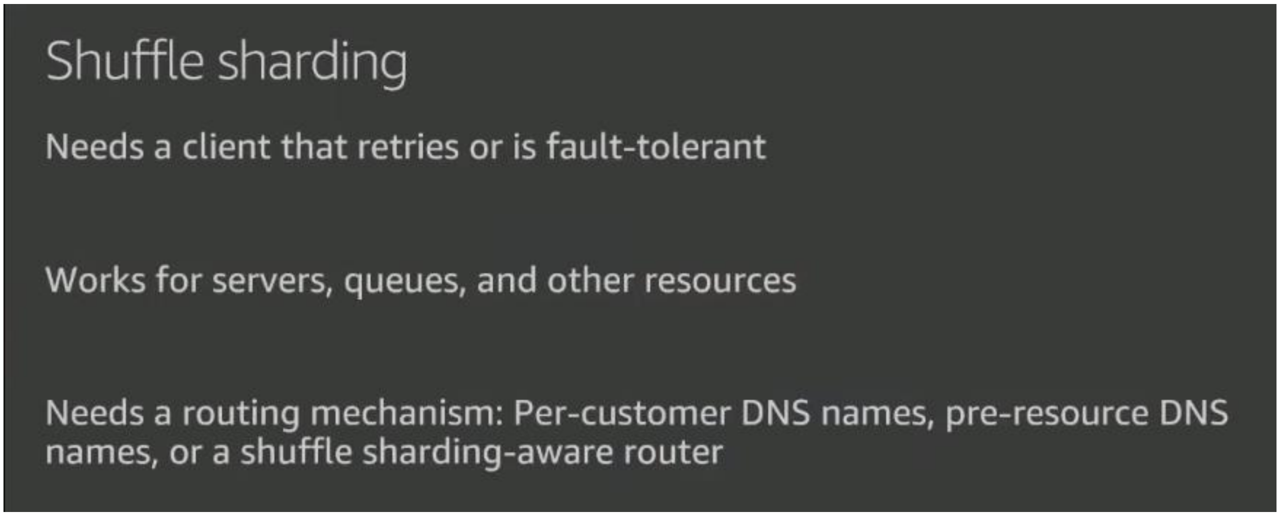
Nodes = 8
Shard size = 2

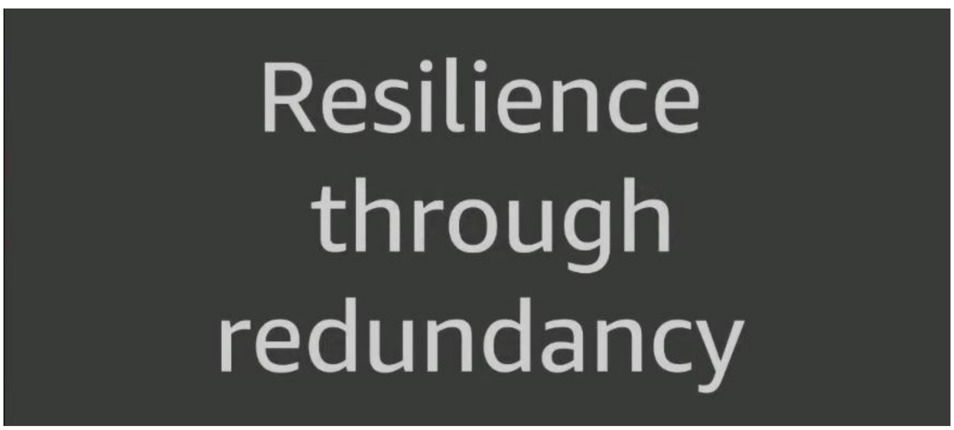
Overlap	Probability
0	53.6%
1	42.8%
2	3.6%

Shuffle sharding

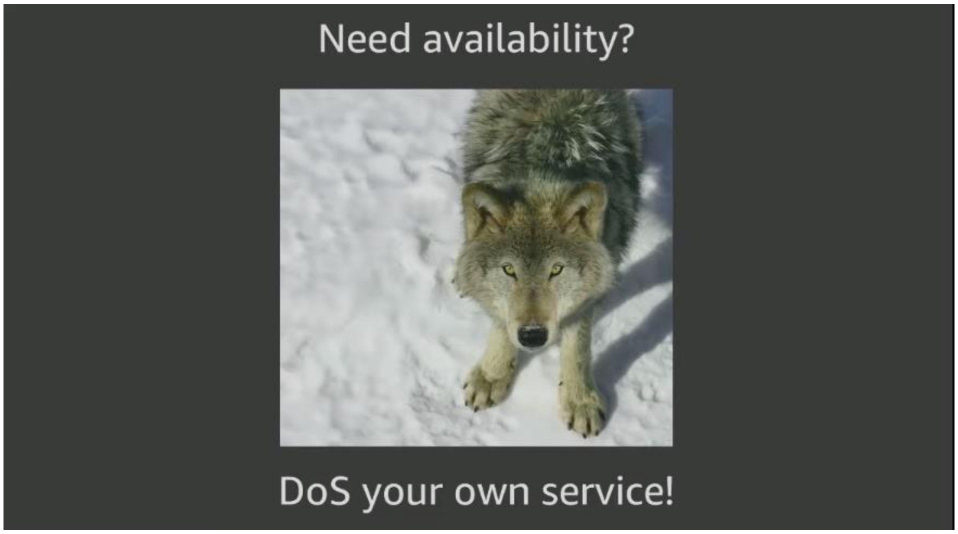
Nodes = 100 Shard size = 5

Overlap	Probability
0	77%
1	21%
2	1.8%
3	0.06%
4	0.0006%
5	0.0000013%



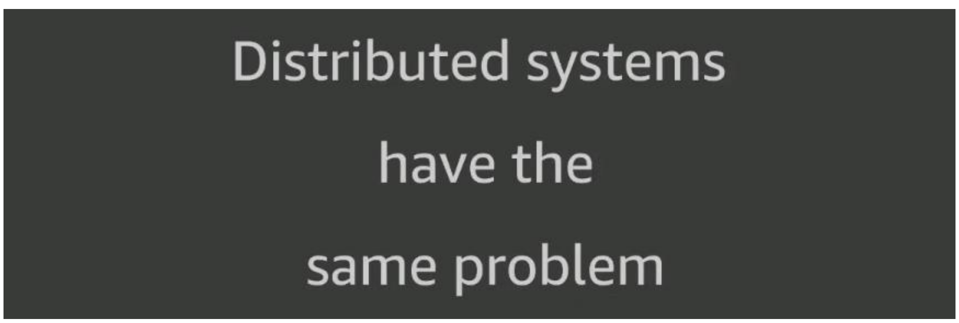


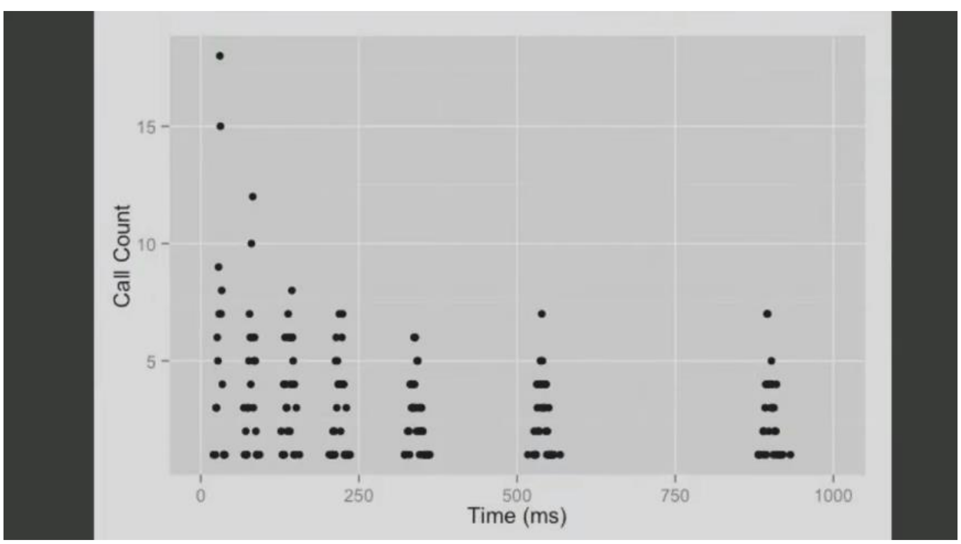
Another challenge is how your clients in a distributed system achieve resilience and HA, by running multiple copies that enable retries when calls to other systems fail



Retries when badly designed can lead to bad consequences

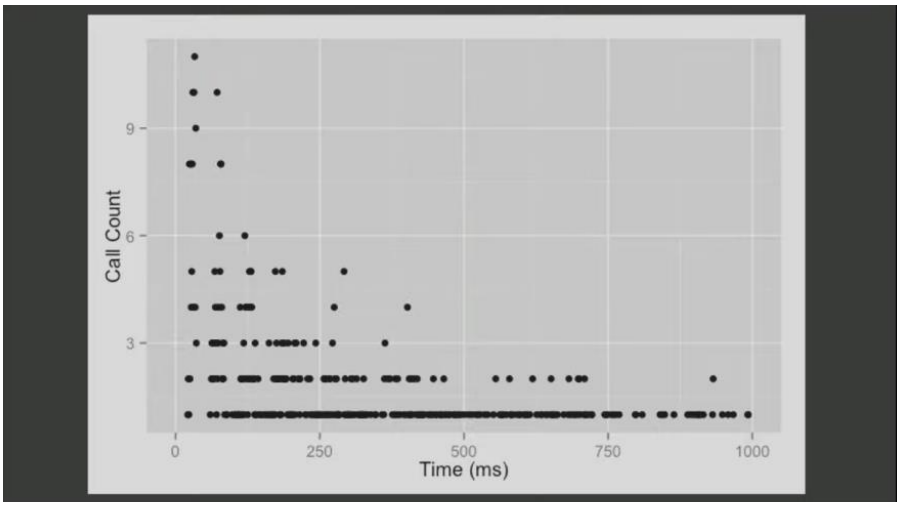
We can defend against that by using exponential back-off retries in our clients or degrade to some stale data.





Humans tend to do things at specific times and this could still lead to DDOS or even cause less utilization of resources during down times

Jitter allows us to introduce randomness into our systems



Doing things at random times is introducing jitter, while not requiring coordination between our systems

```
randint(0, base * 2 ** attempt)
```

This formula can be used to add jitter into our systems

How we do this at AWS

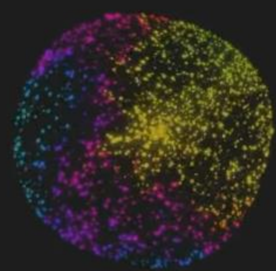
Always jitter when using backoff

Always jitter periodic work (timers, cron, etc.)

Consider adding jitter to all work

Meta lesson: Learn from each other

How does Amazon…



build resiliency into distributed systems?

approach DevOps?

engineer at scale?

The Amazon Builders' Library

How Amazon builds and operates software

- Architecture, software delivery, and operations
- By Amazon's senior technical executives and engineers
- Real-world practices with detailed explanations
- Content available for free on the website

The Amazon Builders' Library

ARCHITECTURE

LEVEL 300

Leader election in distributed systems

Author: Marc Brooker

Improving efficiency, reducing coordination, and simplifying architectures by using leader election.

SOFTWARE DELIVERY AND OPERATIONS

LEVEL 300

Going faster with continuous delivery

Author: Mark Mansour

Automating the software testing and deployment process for speed and reliability

SOFTWARE DELIVERY AND OPERATIONS

LEVEL 400

Implementing health checks

Author: David Yanacek

Automatically detecting and mitigating server failures without unintended consequences from fleet-wide false positives.

ARCHITECTURE

LEVEL 400

Workload isolation using shuffle-sharding

Author: Colm MacCarthaigh

Shuffle Sharding is one of our core techniques for drastically limiting the scope of impact of operational issues

Many more at aws.amazon.com/builders-library

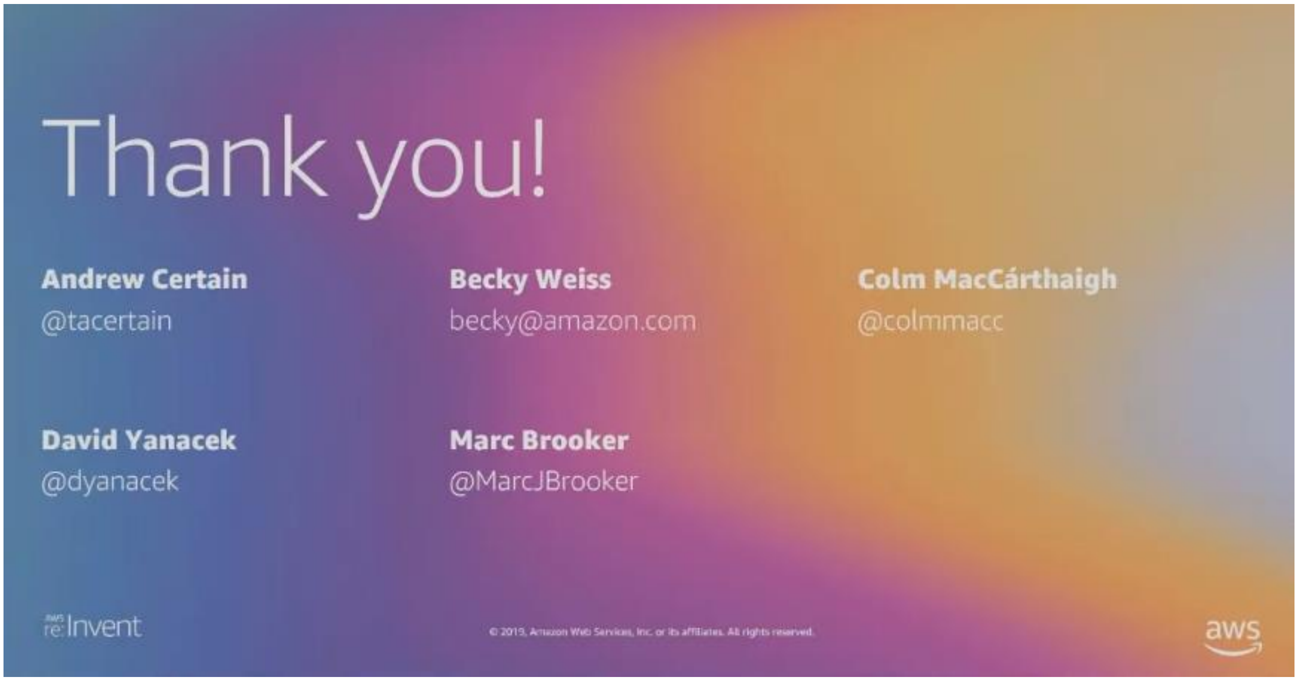
Learn DevOps with AWS Training and Certification

Resources created by the experts at AWS to propel your organization and career forward

- Take free digital training to learn best practices for developing, deploying, and maintaining applications
- Classroom offerings, like DevOps Engineering on AWS, feature AWS expert instructors and hands-on activities
- Validate expertise with the **AWS Certified DevOps Engineer - Professional** or **AWS Certified Developer - Associate** exams

Visit aws.amazon.com/training/path-developing/

aws training and certification


Thank you!
Andrew Certain
@tacertain
Becky Weiss
becky@amazon.com
Colm MacCárthaigh
@colmmacc
David Yanacek
@dyanacek
Marc Brooker
@MarcJBrooker
re:Invent
© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
aws