Twitter Thread by Yan Cui is making the AppSync Masterclass

Yan Cui is making the AppSync Masterclass
@theburningmonk

If you missed @clare_liguori's Continuous Delivery session this week (like I did) then good news, it's available on-demand now ■

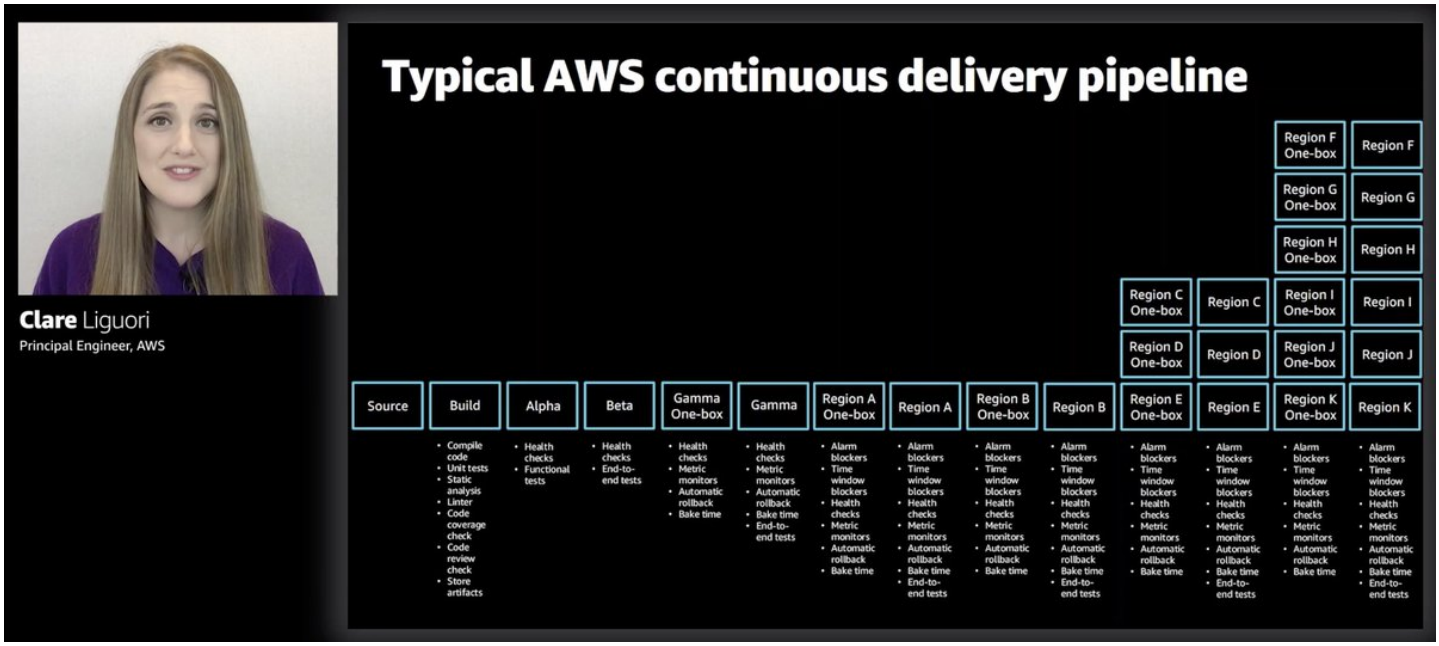
https://t.co/78b8sI2hHs

And here's my play-by-play for the session

■...

This is a typical CD pipeline in AWS.

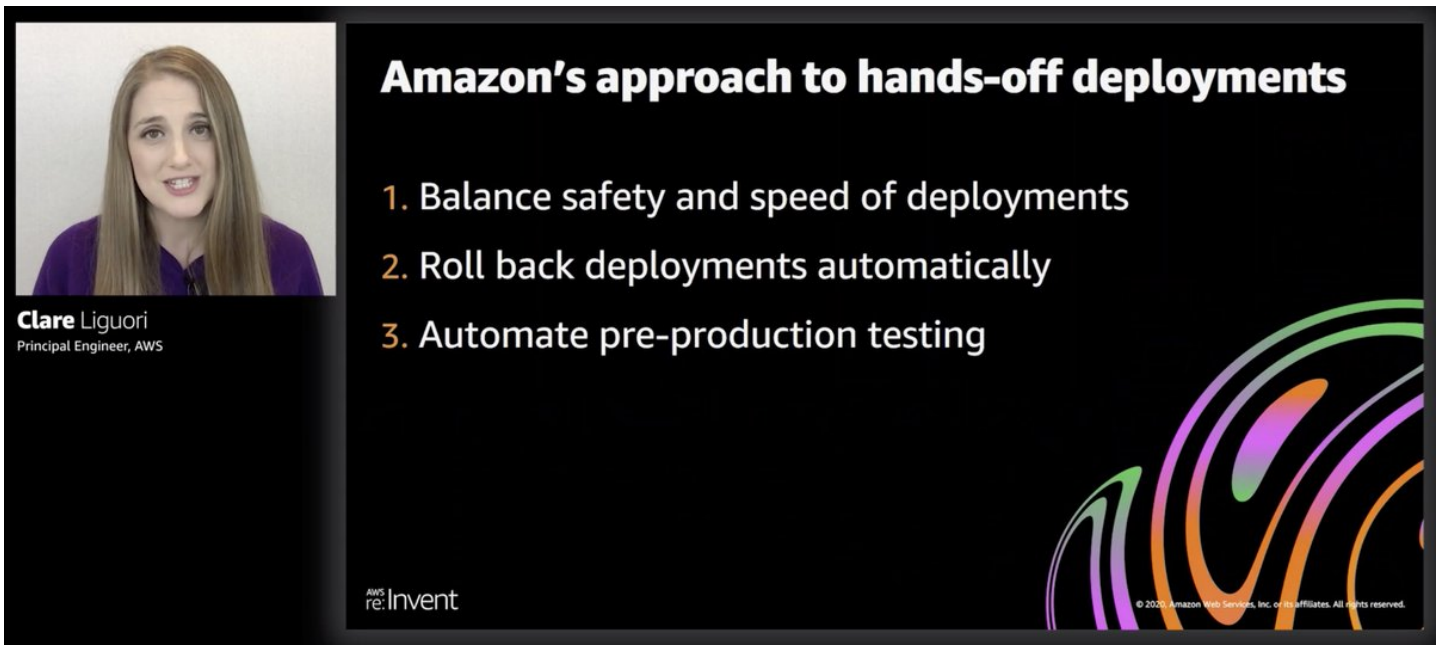
This is far more complex than the most complex CD pipeline I have ever had! Just cos it's complex, doesn't mean it's over-engineered though. Given the blast radius, I'm glad they do releases carefully and safely.



If you look closely, beyond all the alpha, beta, gamma environments, it's one-box in a region first then the rest of the region, I assume starting with the least risky regions first.

For anyone thinking about going multi-region (after the recent Kinesis outage), this is one of the complexities you need to consider. To do multi-region right and deploy safely (minimize blast radius), this is one of the complexities you have to factor in.

This "deploy small at first then more broadly" principle applies to #serverless apps too, though you can't "deploy to one box". You can do it with canary deployments instead, CodeDeploy supports this practice for Lambda (using weighted aliases) out-of-the-box.



However, weighted-alias has no session-affinity and it's not possible to propagate the canary decision along the call chain (e.g. when an API function invokes another function via SNS/EventBridge, etc.)...

More details in this post:
https://t.co/l0f44tunJD

This problem applies to API Gateway's canary support too, which has no session affinity, so a user making 2 requests (for a paginated endpoint) can yo-yo between the canary and current production channels.

For simple use cases, this might be fine, but hardly ideal for minimizing blast radius, or if you want to do some A/B tests on new features.

Personally, I love what you can do with @LaunchDarkly such a slick control panel and super easy to use ❤■

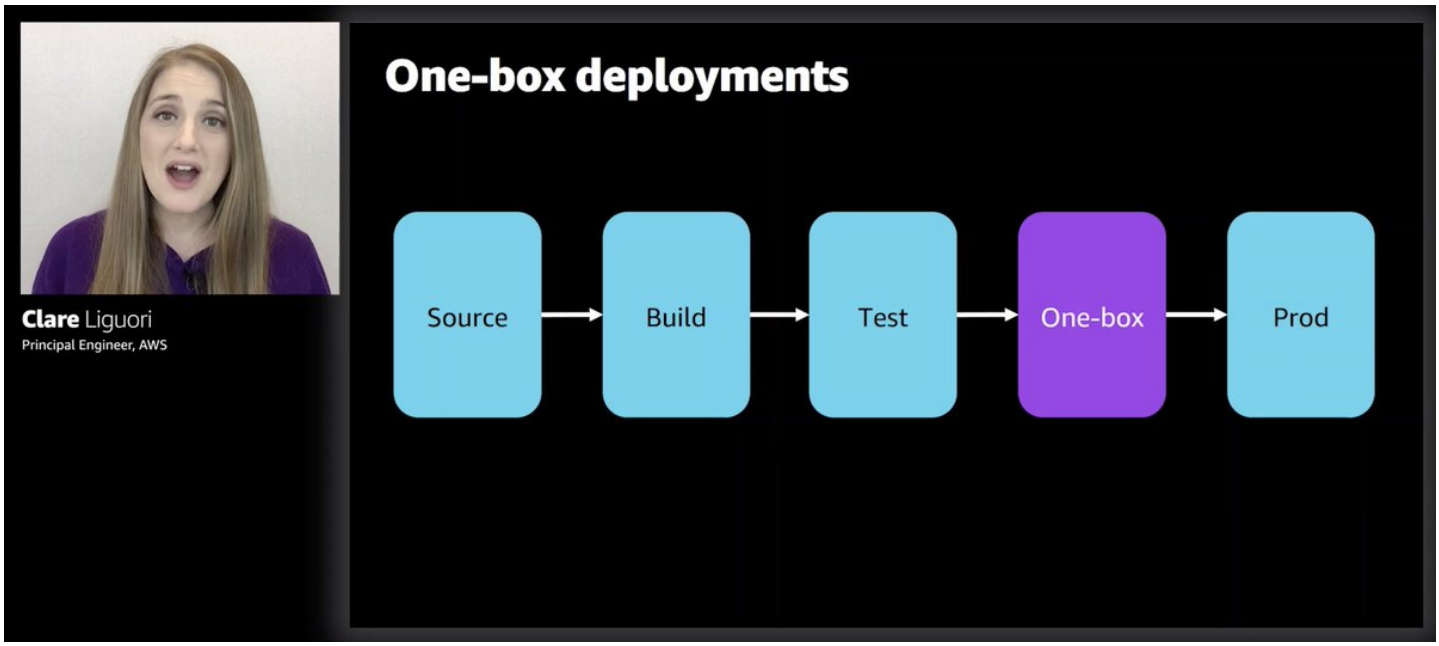
But for Lambda functions, it gets a bit trickier because you need so many persistent connections... so, your best bet is to use a proxy (run it in Fargate) as I described in this post: https://t.co/O0W62mU2AN

It can get expensive though, because you're hitting DynamoDB a lot!

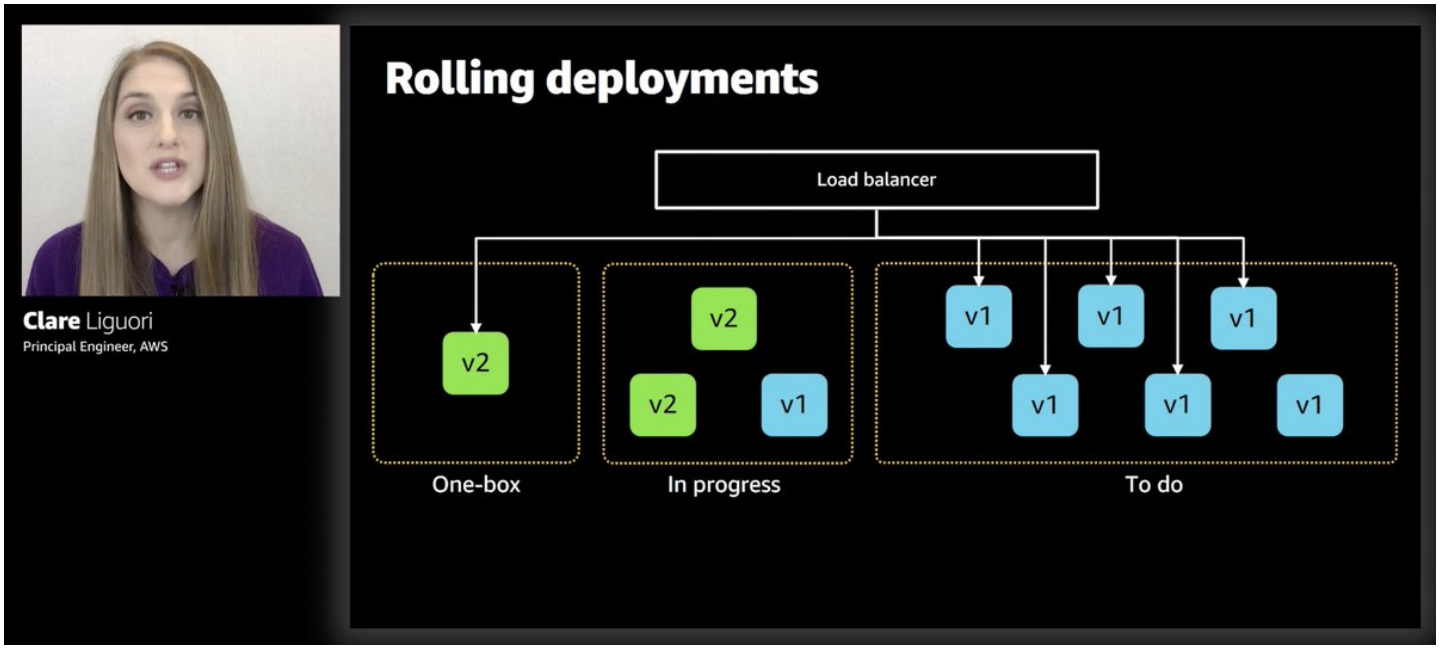
Anyway, I digress...

"One box used to mean one VM, but over time it has also come to mean one container or a small percentage of Lambda function invocations"

ha, so they use weighted-alias for microservices that run on Lambda too, starting at 10% at first

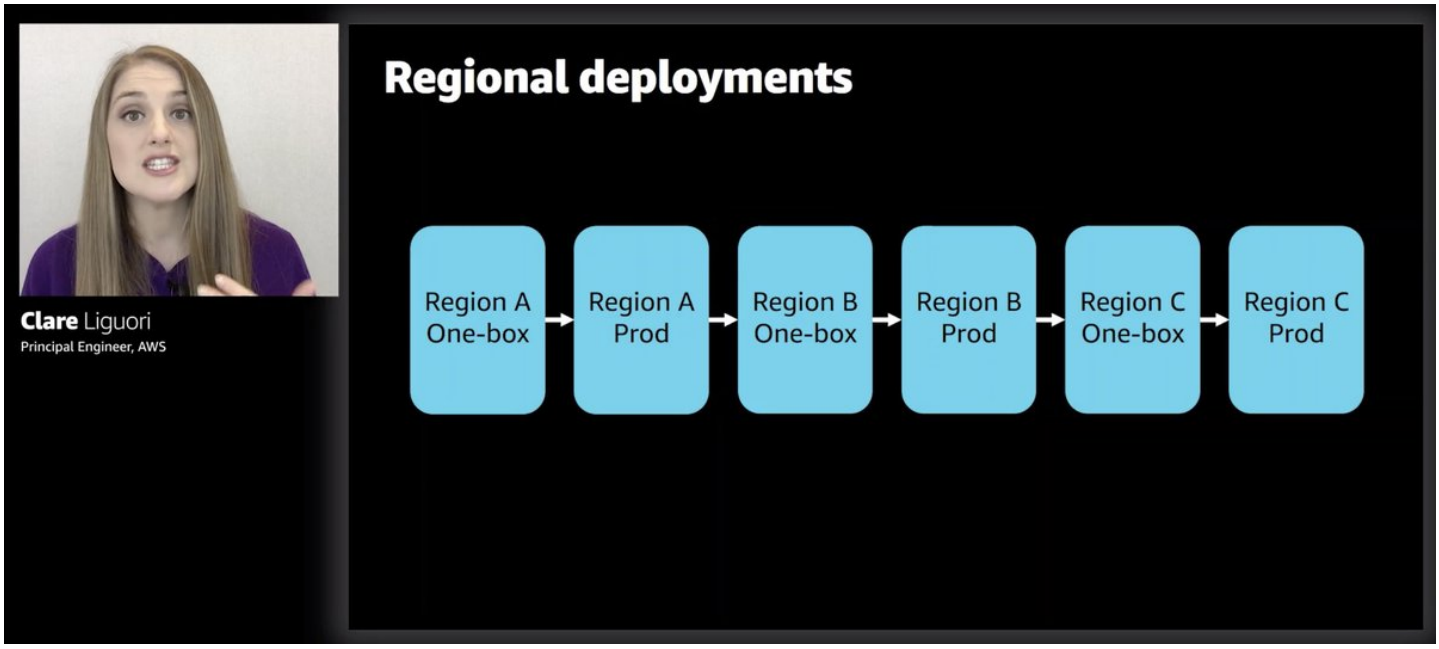


And instead of rolling out the other 90% all at once (which is still risky), they use rolling deployment, which, CodeDeploy supports also.



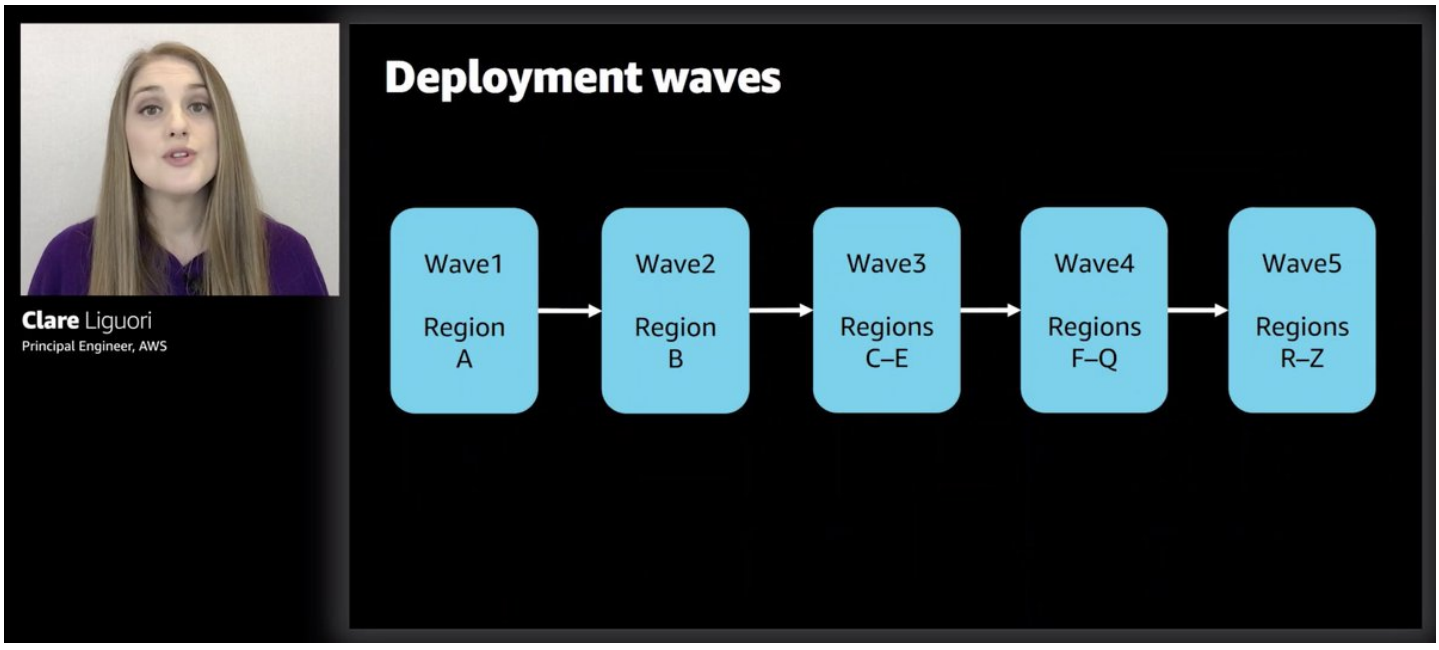
Do this "one-box => rolling deploy pattern to the rest" pattern in one region first, then rinse and repeat for the other regions.

And within each region, apply the same pattern to AZs too.



To crawl back some speed (otherwise, every deployment would take weeks...) they deploy the regions in waves.

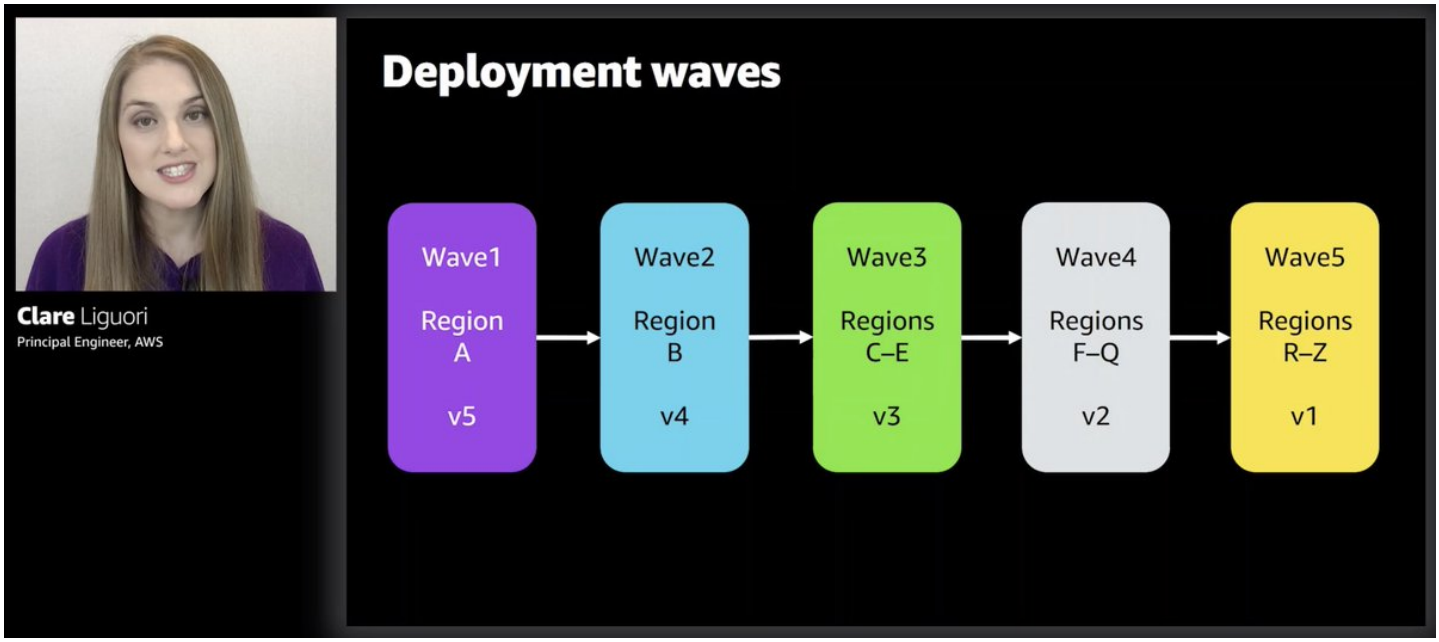
First few waves deploy to one region and one AZ at a time, later waves (after you build some confidence) deploy to multiple regions in parallel



mm.. this is interesting!

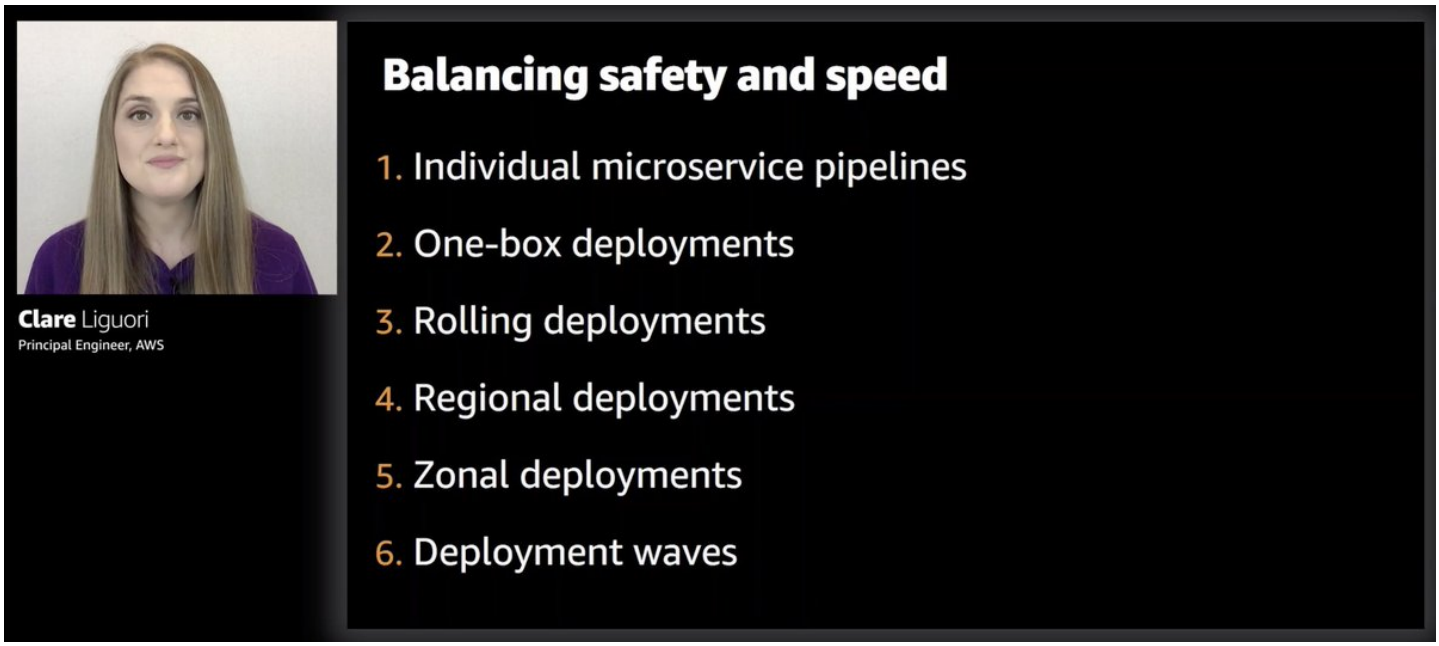
Deployments are staggered, so multiple deployments can be in motion and at different stages at once.

How does this affect rollback I wonder ■ e.g. if v1 deployment craps out at wave 5 and triggers rollback, what of wave 1 which is deploying v5?



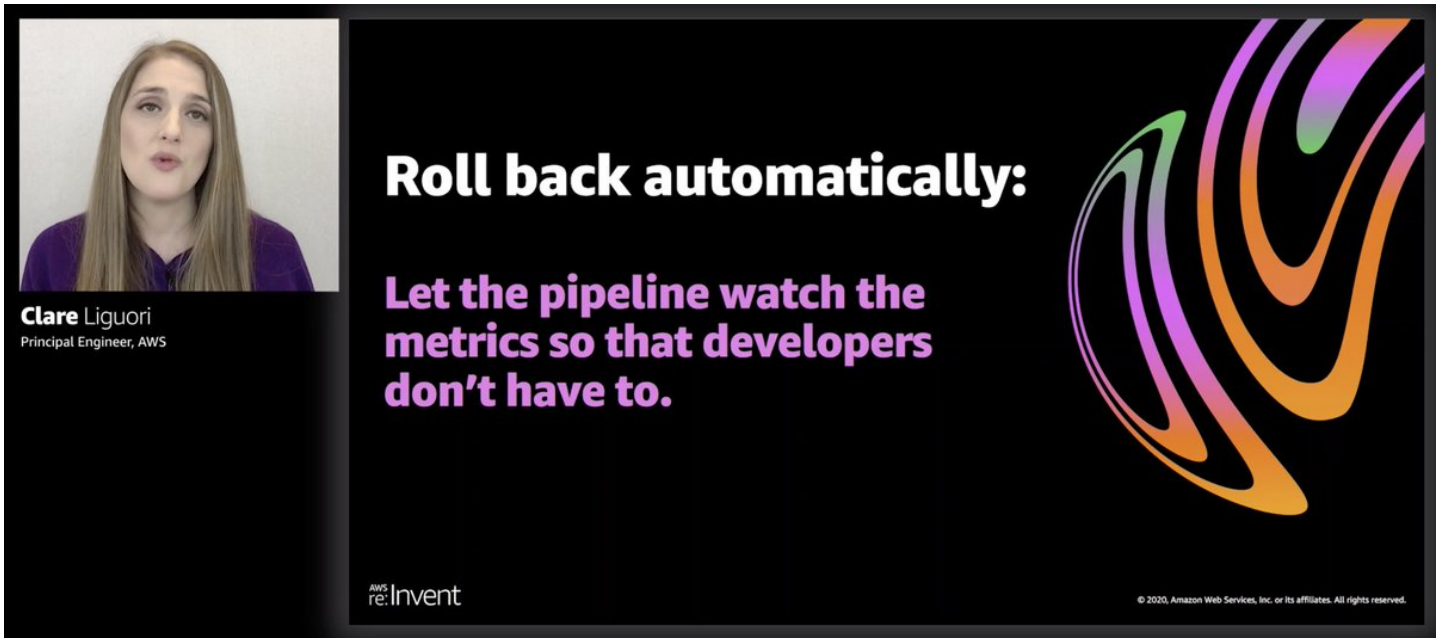
I had to build custom mechanisms to stop parallel deployments in the past, because of the complication to rollbacks. Interesting to see AWS had gone the other way. But I get why they do it, to get some speed back.

Summary for this section of the talk. Automatic rollback next, really interested to see how that works with respect to these staggered deployment waves.

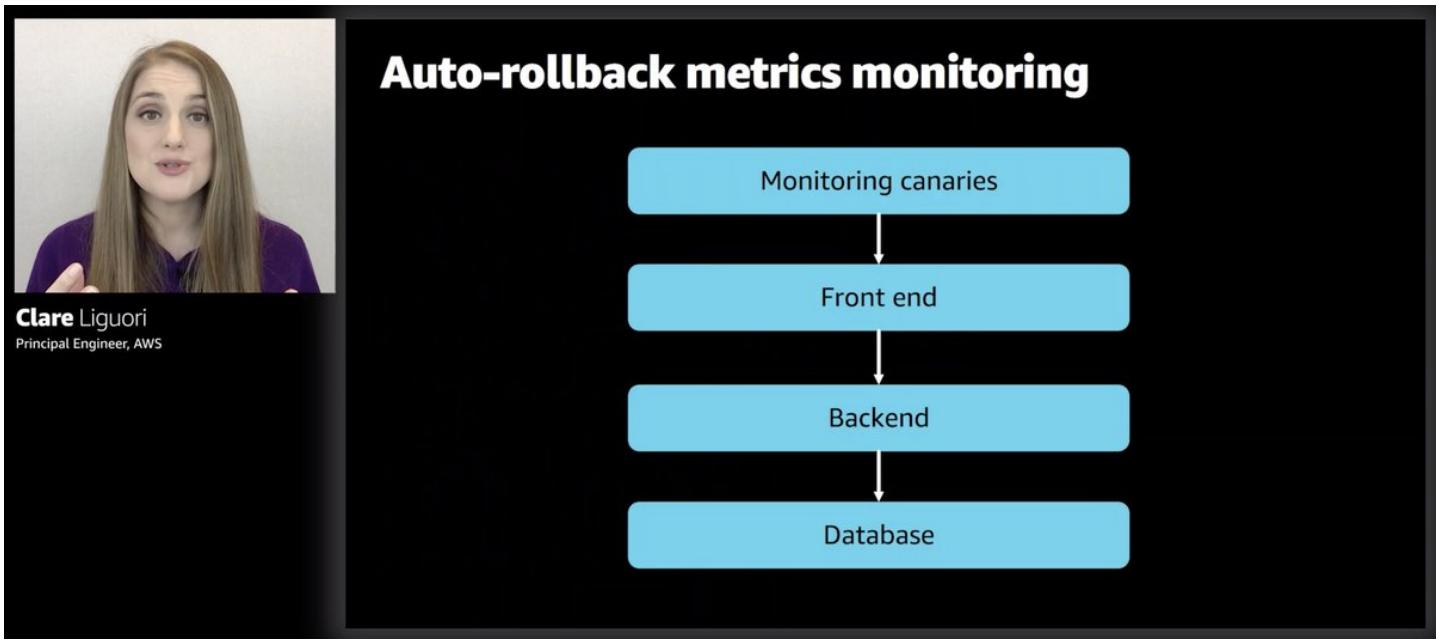


"At Amazon, we don't want to have to sit and stare at the dashboard every time we do a deployment, we want to deployments to be hands-off"

Have thresholds on a bunch of metrics (regional, zonal aggregates as well as per-box) to trigger automatic rollbacks.

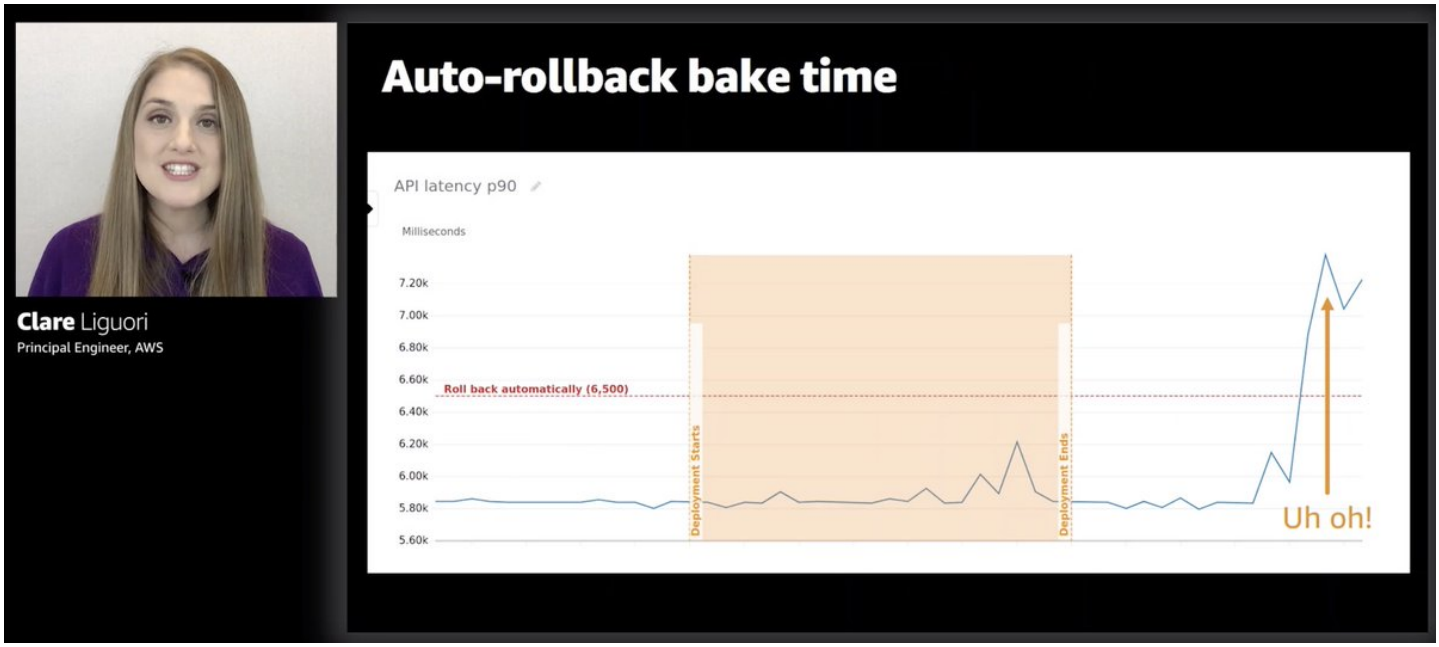


And they also use monitoring canaries (which, as an AWS customer, we have CloudWatch Synthetics for that) so they can look at system health more holistically to trigger rollback.

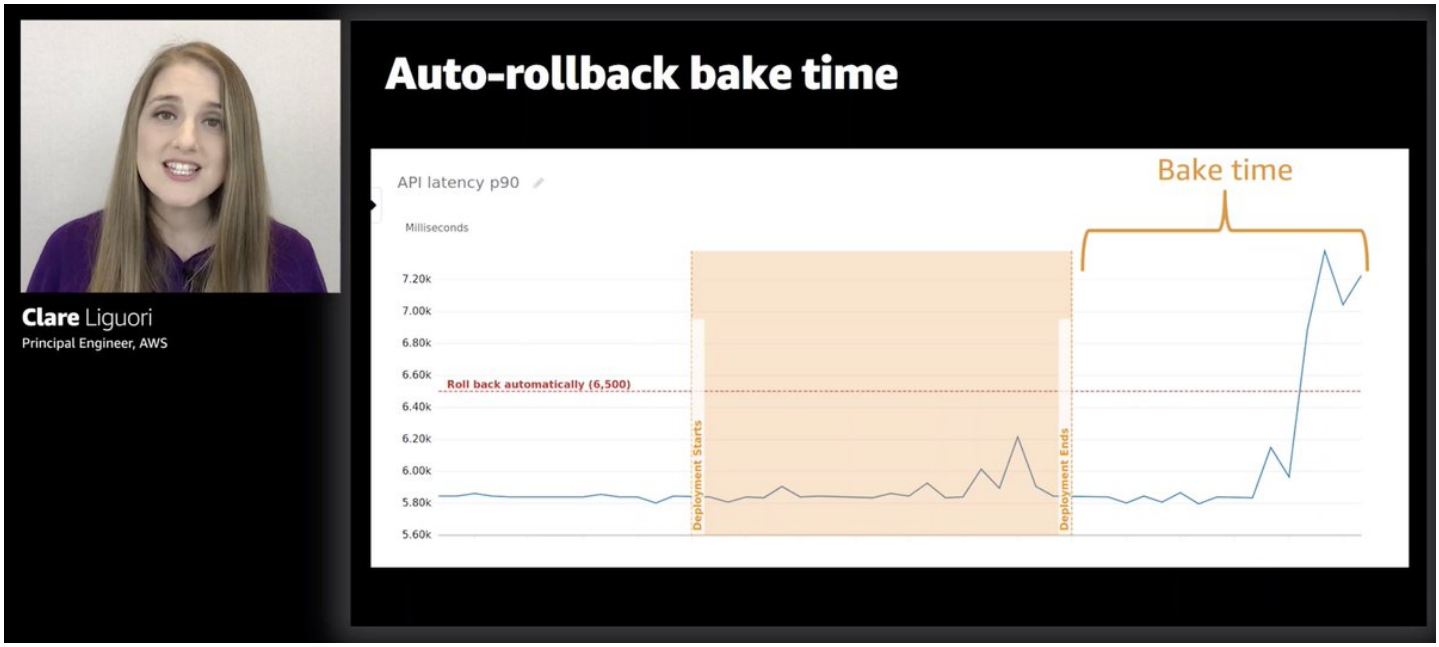


"The impact from a deployment doesn't always show up during a deployment"

haha, been there... once had a slow memory leak that showed up 2 weeks after a deployment ■■■■



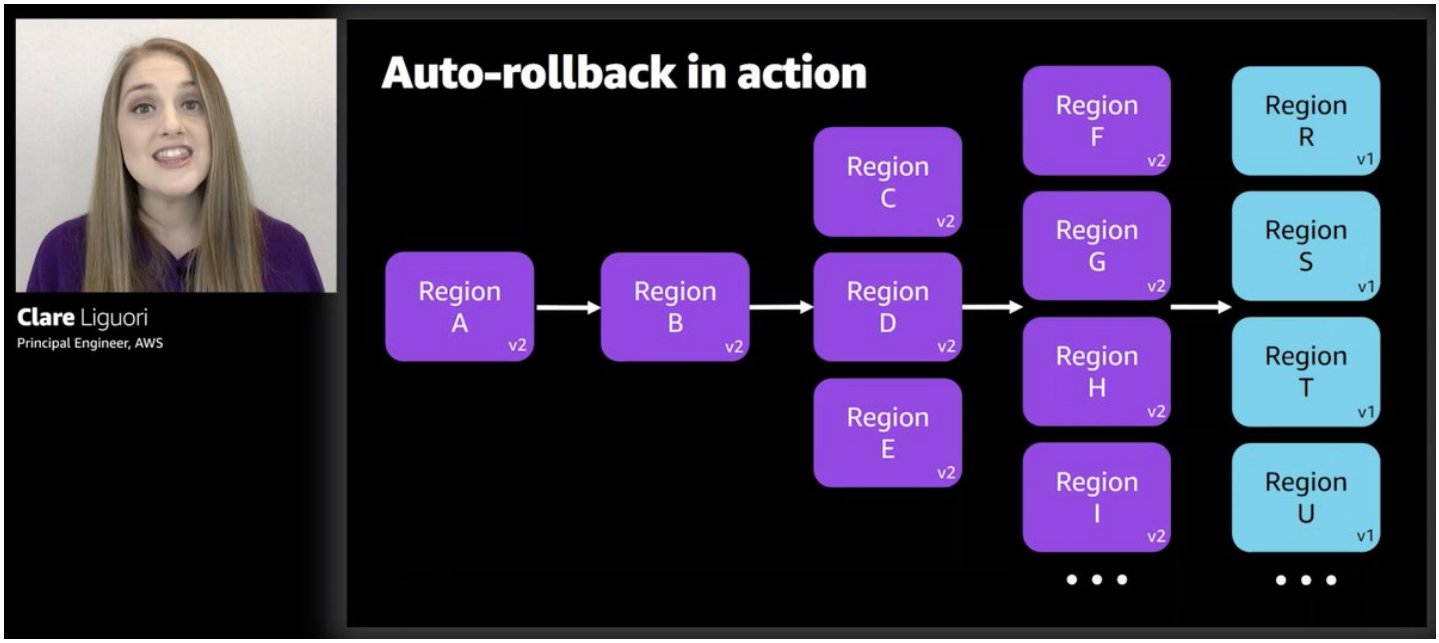
The pipeline continues to monitor the metrics during bake time for a deployment. And it'll hold the deployment during the bake time, and not let the deployment move onto the next stage under after the bake time. Otherwise, you can be seeing the impact of another deployment.



Finally, here it is:

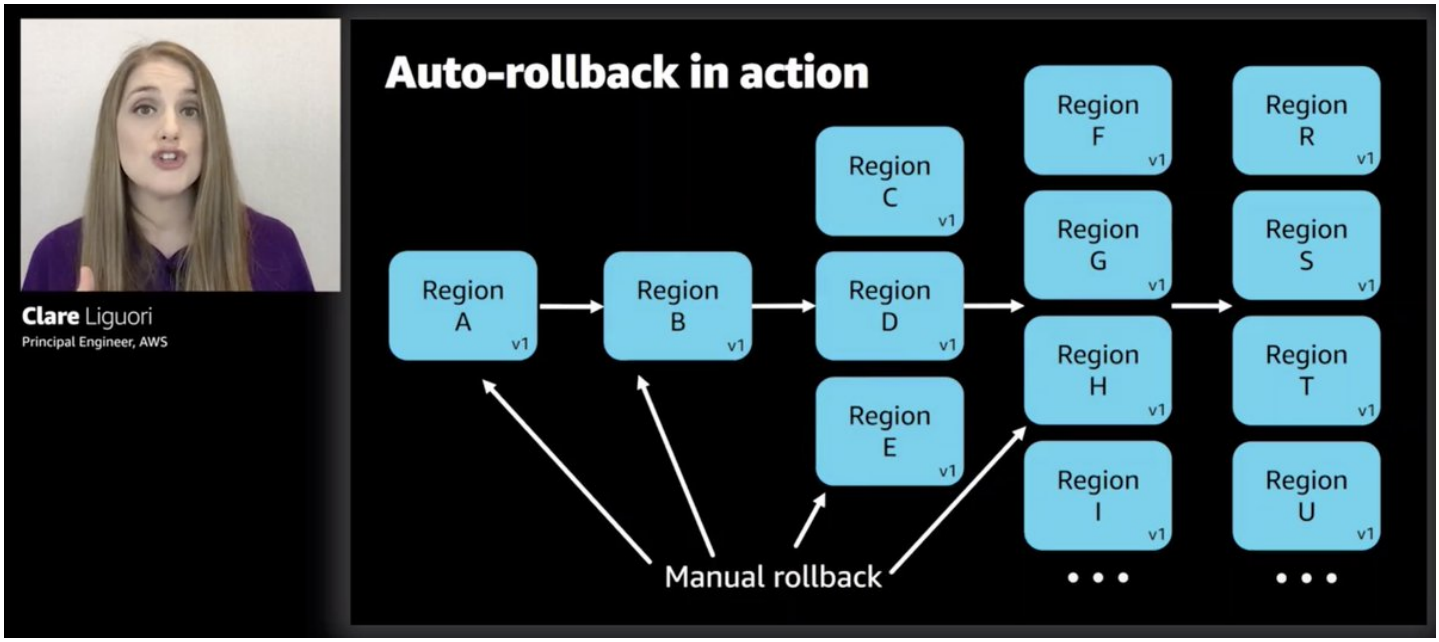
1. auto-rollback only in regions/zones that tripped the threshold
2. eng has to decide whether to roll back the whole thing or retry

e.g. something else could have happened in the offending region, which is why thresholds were crossed



If they decide to rollback, then everything gets rolled back.

Or, they can roll forward and push out a v3 deployment instead, which fixes whatever problem that got picked up in that failed region.

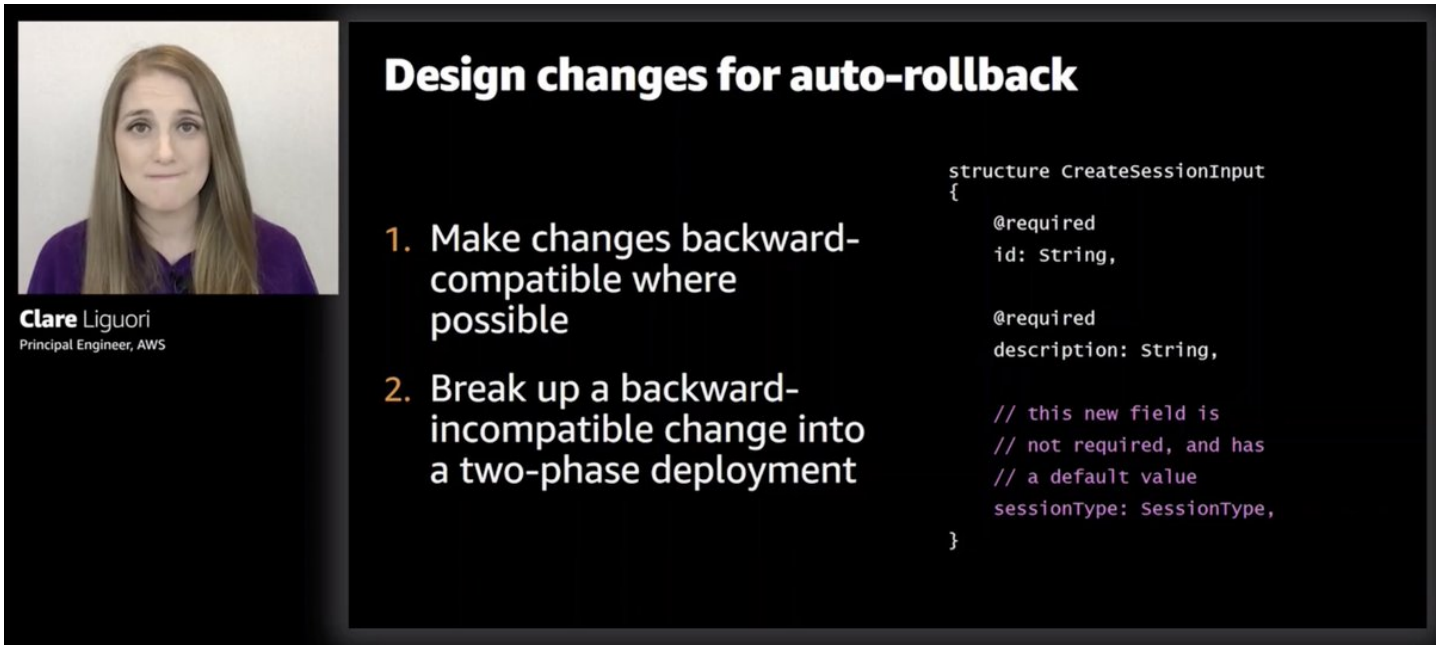


In the scenario where you have v2 and v3 deployments happening at the same time (at different stages), if v2 hit a snag and has to rollback the whole thing. I wonder if they'd rollback any v3 changes that's been applied to the regions in earlier waves too. ■

That seems the only sensible thing to do.

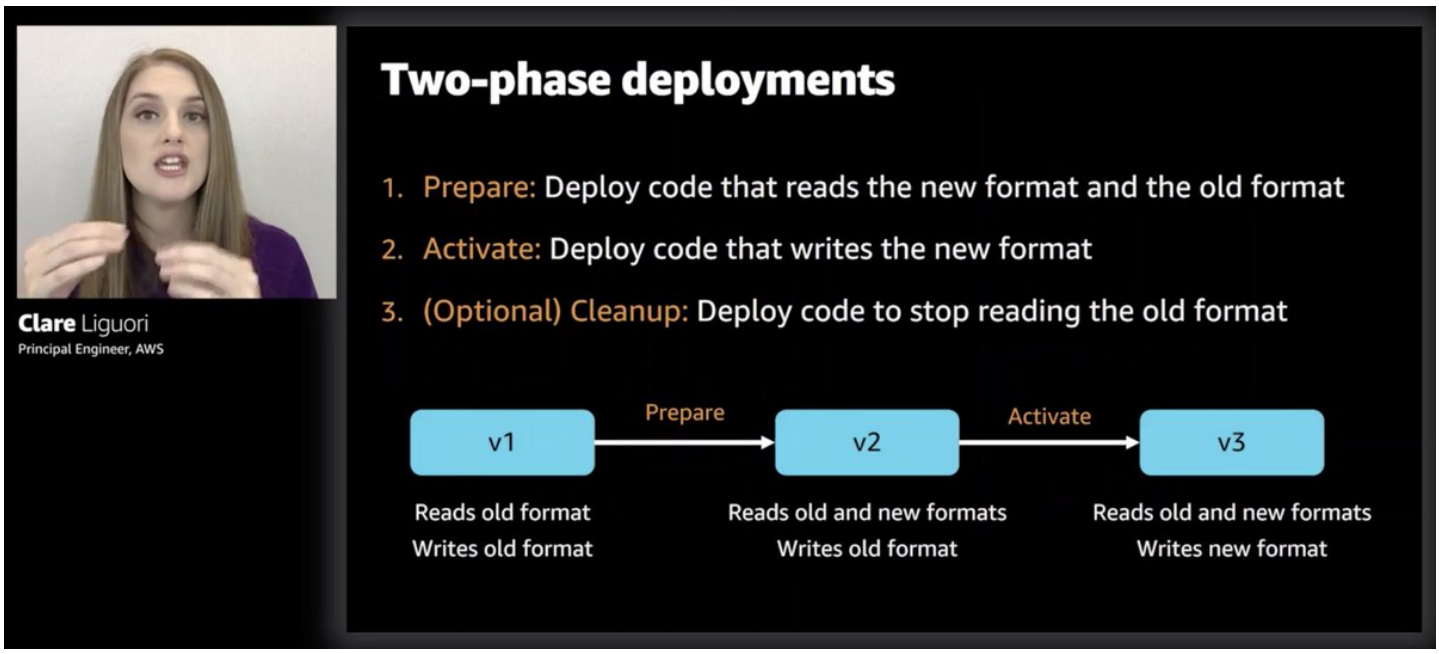
Anyway, Claire moves onto how to design your changes so they can be rolled back automatically.

As much as possible, make backward-compatible changes ■



Otherwise, you're forced to make a phased deployment where each phase contains backward-compatible changes.

This mirrors a lot of database migrations when you move from one database to another one and you can't do it with downtime.

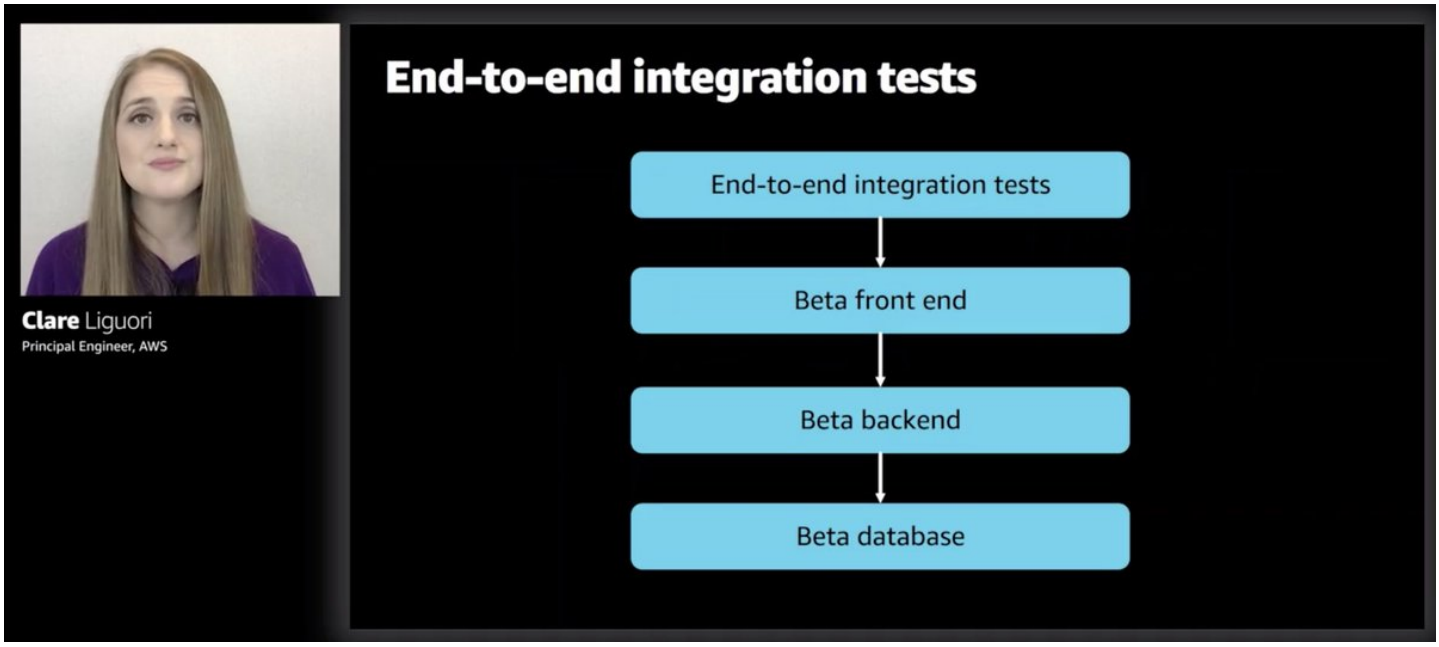


Seriously though, if you need to make breaking changes, first see if you can do it with a small downtime. It'll save you so much complexity and extra work.

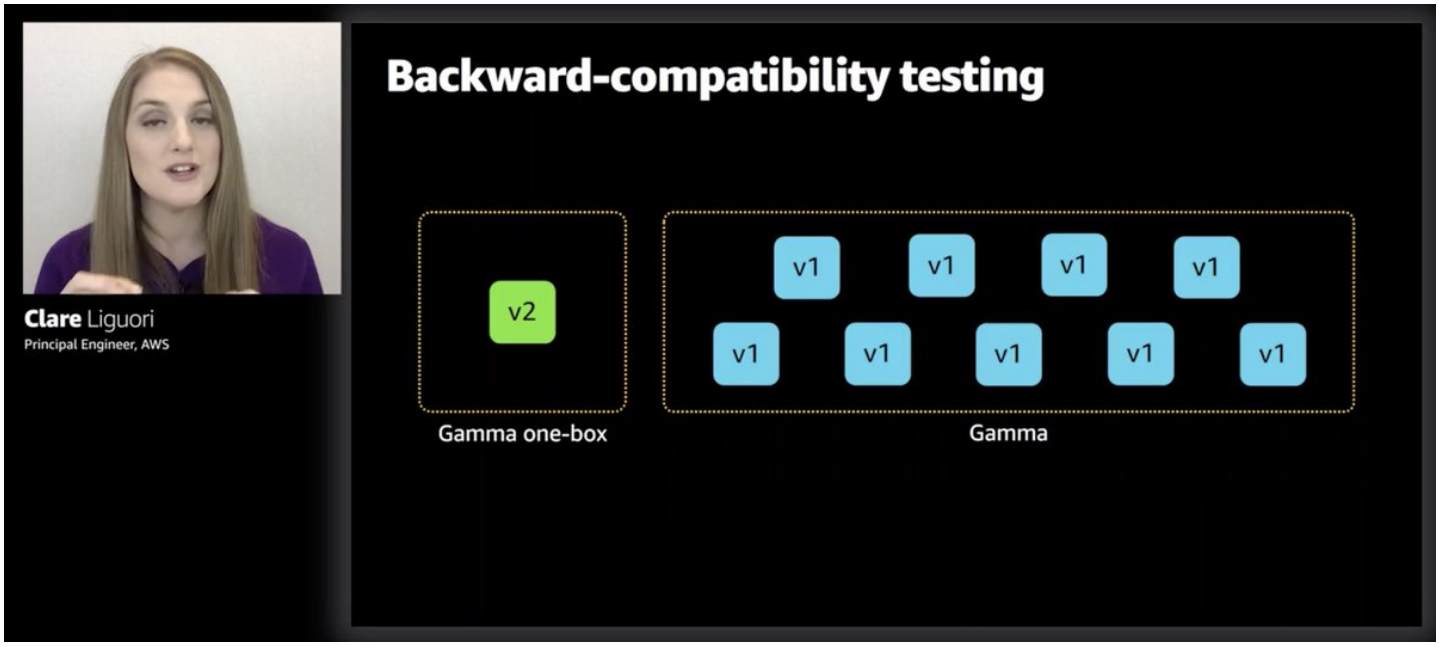
It's not an option at AWS scale of course, but you're not AWS.

Before they even get to the production deployment, there's a bunch of pre-production test environments.

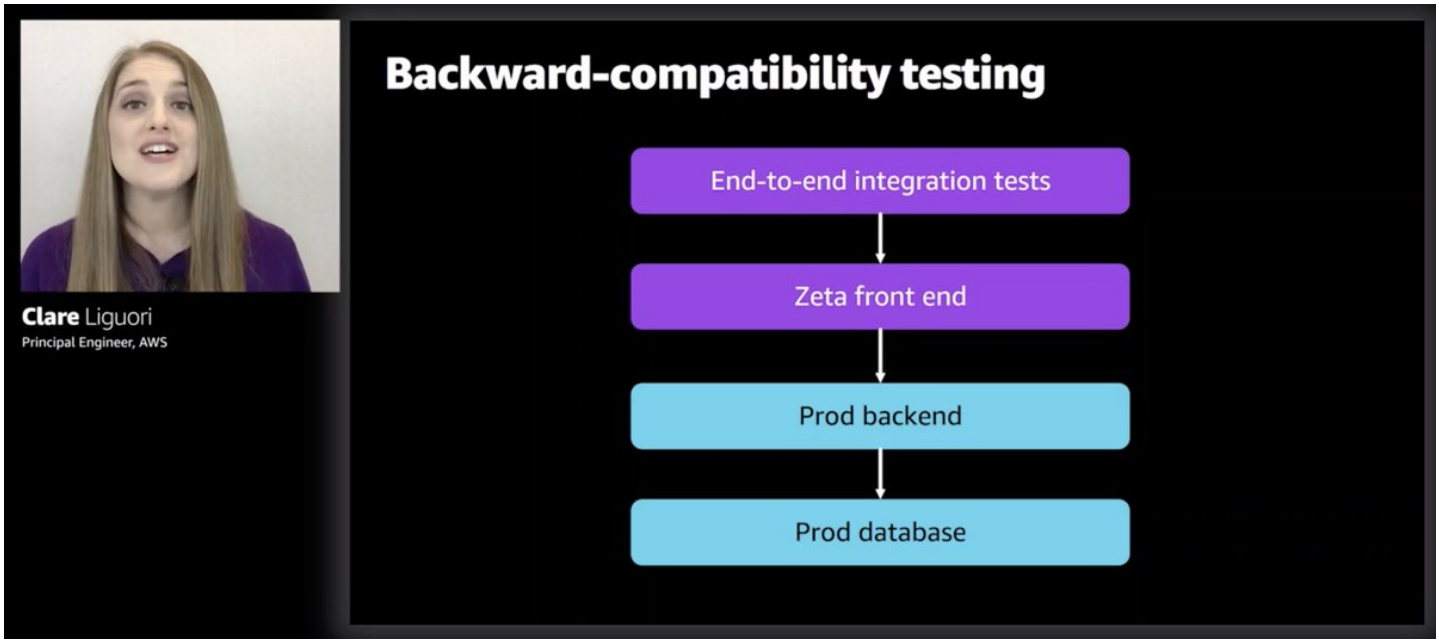
And they basically practice the one-box deployment in the gamma (production-like) environment.



The one-box deployment in Gamma gives them a bit of backward-compatibility test, that it's ok for there to be two versions running side-by-side. So the monitoring canaries would help pick up incompatibilities there.



Some teams go even further with backward-compatibility test by adding another zeta stage to make sure new frontend works with current production backend.



That was great. So nice to see what AWS is doing to ensure deployments are safe and fast (well, as fast as can be without putting customers at risk)

If you wanna catch the session yourself, here's the on-demand video: https://t.co/78b8sI2hHs

