Advancing safe deployment practices

Posted on February 5, 2020

Mark Russinovich
Chief Technology Officer and Technical Fellow, Microsoft Azure

"What is the primary cause of service reliability issues that we see in Azure, other than small but common hardware failures? Change. One of the value propositions of the cloud is that it's continually improving, delivering new capabilities and features, as well as security and reliability enhancements. But since the platform is continuously evolving, change is inevitable. This requires a very different approach to ensuring quality and stability than the box product or traditional IT approaches — which is to test for long periods of time, and once something is deployed, to avoid changes. This post is the fifth in the series (https://azure.microsoft.com/en-us/blog/tag/advancing-reliability/) I kicked off in my July blog post (https://azure.microsoft.com/en-us/blog/advancing-microsoft-azure-reliability/) that shares insights into what we're doing to ensure that Azure's reliability supports your most mission critical workloads. Today we'll describe our **safe deployment practices**, which is how we manage change automation so that all code and configuration updates go through well-defined stages to catch regressions and bugs before they reach customers, or if they do make it past the early stages, impact the smallest number possible. **Cristina del Amo Casado** from our Compute engineering team authored this posts, as she has been driving our safe deployment initiatives." - Mark Russinovich, CTO, Azure

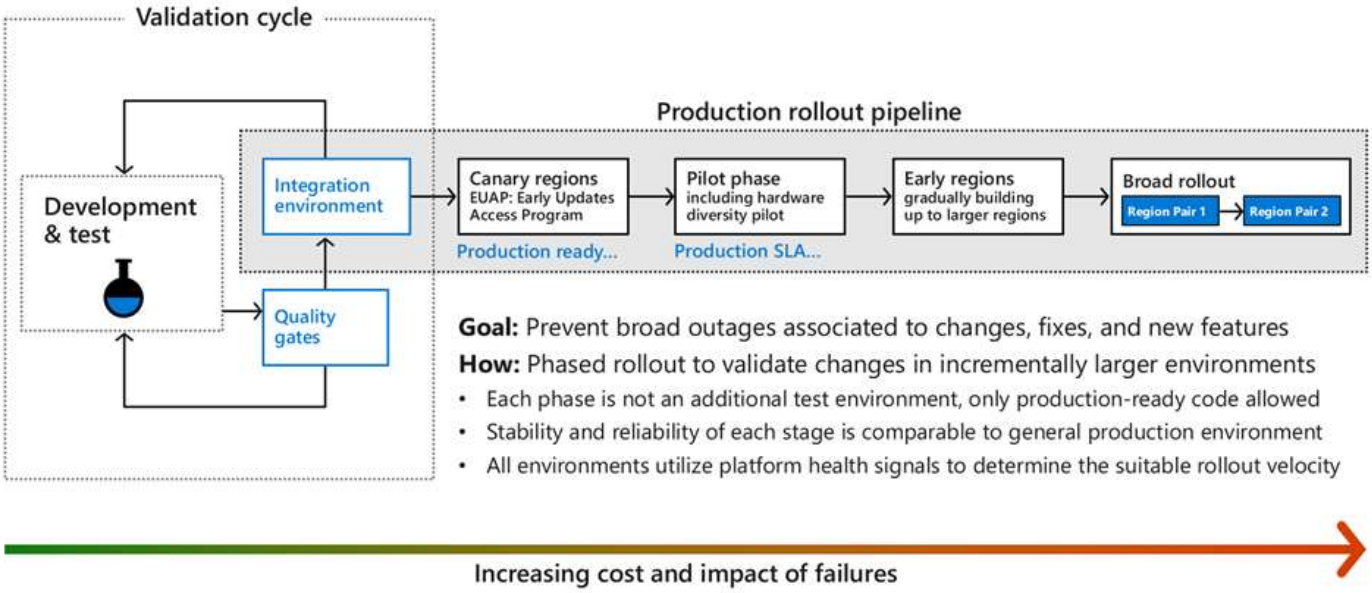
When running IT systems on-premises, you might try to ensure perfect availability by having gold-plated hardware, locking up the server room and throwing away the key. Software wise, IT would traditionally prevent as much change as possible — avoiding applying updates to the operating system or applications because they're too critical, and pushing back on change requests from users. With everyone treading carefully around the system, this 'nobody breathe!' approach stifles continued system improvement, and sometimes even compromises security for systems that are deemed too crucial to patch regularly. As Mark mentioned above, this approach doesn't work for change and release management in a hyperscale public cloud like Azure. Change is both inevitable and beneficial, given the need to deploy service updates and improvements, and given our commitment to you to act quickly in the face of security vulnerabilities. As we can't simply avoid change, Microsoft, our customers, and our partners need to acknowledge that change is expected, and we plan for it. Microsoft continues to work on making updates as transparent as possible and will deploy the changes safely as described below. Having said that, our customers and partners should also design for high availability, consume maintenance events sent by the platform to adapt as needed. Finally, in some cases, customers can take control of initiating the platform updates at a suitable time for their organization (https://www.aka.ms/advancingreliability/4).

Changing safely

When considering how to deploy releases throughout our Azure datacenters, one of the key premises that shapes our processes is to assume that there could be an unknown problem introduced by the change being deployed, plan in a way that enables the discovery of said problem with minimal impact, and automate mitigation actions for when the problem surfaces. While a developer might judge it as completely innocuous and guarantee that it won't affect the service, even the smallest change to a system poses a risk to the stability of the system, so 'changes' here refers to all kinds of new releases and covers both code changes and configuration changes. In most cases a configuration change has a less dramatic impact on the behavior of a system but, just as for a code change, no configuration change is free of risk for activating a latent code defect or a new code path.

Teams across Azure follow similar processes to prevent or at least minimize impact related to changes. Firstly, by ensuring that changes meet the quality bar before the deployment starts, through test and integration validations. Then after sign off, we roll out the change in a gradual manner and measure health signals continuously, so that we can detect in relative isolation if there is any unexpected impact associated with the change that did not surface during testing. We do not want a change causing problems to ever make it to broad production, so steps are taken to ensure we can avoid that whenever possible. The gradual deployment gives us a good opportunity to detect issues at a smaller scale (or a smaller 'blast radius') before it causes widespread impact.

Azure approaches change automation, aligned with the high level process above, through a **safe deployment practice (SDP) framework**, which aims to ensure that all code and configuration changes go through a lifecycle of specific stages, where health metrics are monitored along the way to trigger automatic actions and alerts in case of any degradation detected. These stages (shown in the diagram that follows) reduce the risk that software changes will negatively affect your existing Azure workloads.



(https://azurecomcdn.azureedge.net/mediahandler/acomblog/media/Default/blog/dfe1ed0f-f1f3-45c7-bd0d-9cb934b15eed.png)

This shows a simplification of our deployment pipeline, starting on the left with developers modifying their code, testing it on their own systems, and pushing it to staging environments. Generally, this **integration environment** is dedicated to teams for a subset of Azure services that need to test the interactions of their particular components together. For example, core infrastructure teams such as compute, networking, and storage share an integration environment. Each team runs synthetic tests and stress tests on the software in that environment, iterate until stable, and then once the quality results indicate that a given release, feature, or change is ready for production they deploy the changes into the canary regions.

Canary regions

Publicly we refer to **canary regions** as "Early Updates Access Program" regions, and they're effectively full-blown Azure regions with the vast majority of Azure services. One of the canary regions is built with Availability Zones (https://docs.microsoft.com/en-us/azure/availability-zones/az-overview) and the other without it, and both regions form a region pair (https://docs.microsoft.com/en-us/azure/best-practices-availability-paired-regions) so that we can validate data geo-replication capabilities. These canary regions are used for full, production level, end to end validations and scenario coverage at scale. They host some first party services (for internal customers), several third party services, and a small set of external customers that we invite into the program to help increase the richness and complexity of scenarios covered, all to ensure that canary regions have patterns of usage representative of our public Azure regions. Azure teams also run stress and synthetic tests in these environments, and periodically we execute fault injections or disaster recovery drills at the region or Availability Zone level, to practice the detection and recovery workflows that would be run if this occurred in real life. Separately and together, these exercises attempt to ensure that software is of the highest quality before the changes touch broad customer workloads in Azure.

Pilot phase

Once the results from canary indicate that there are no known issues detected, the progressive deployment to production can get started, beginning with what we call our **pilot phase**. This phase enables us to try the changes, still at a relatively small scale, but with more diversity of hardware and configurations. This phase is especially important for software like core storage services and core compute infrastructure services, that have hardware dependencies. For example, Azure offers servers with GPU's, large memory servers, commodity servers, multiple generations and types of processors, Infiniband, and more, so this enables flighting the changes and may enable detection of issues that would not surface during the smaller scale testing. In each step along the way, thorough health monitoring and extended 'bake times' enable potential failure patterns to surface, and increase our confidence in the changes while greatly reducing the overall risk to our customers.

Once we determine that the results from the pilot phase are good, the deployment systems proceed by allowing the change to progress to **more and more regions incrementally**. Throughout the deployment to the broader Azure regions, the deployment systems endeavor to respect Availability Zones (a change only goes to one Availability Zone within a region) and region pairing (every region is 'paired up' with a second region for georedundant storage) so a change deploys first to a region and then to its pair. In general, the changes deploy only as long as no negative signals surface.

Safe deployment practices in action

Given the scale of Azure globally, the entire rollout process is completely automated and driven by policy. These declarative policies and processes (not the developers) determine how quickly software can be rolled out. Policies are defined centrally and include mandatory health signals for monitoring the quality of software as well as mandatory 'bake times' between the different stages outlined above. The reason to have software sitting and baking for different periods of time across each phase is to make sure to expose the change to a full spectrum of load on that service. For example, diverse organizational users might be coming online in the morning, gaming customers might be coming online in the evening, and new virtual machines (VMs) or resource creations from customers may occur over an extended period of time.

Global services, which cannot take the approach of progressively deploying to different clusters, regions, or service rings, also practice a version of progressive rollouts in alignment with SDP. These services follow the model of updating their service instances in multiple phases, progressively deviating traffic to the updated instances through Azure Traffic Manager. If the signals are positive, more traffic gets deviated over time to updated instances, increasing confidence and unblocking the deployment from being applied to more service instances over time.

Of course, the Azure platform also has the ability to deploy a change simultaneously to all of Azure, in case this is necessary to mitigate an extremely critical vulnerability. Although our safe deployment policy is mandatory, we can choose to accelerate it when certain emergency conditions are met. For example, to release a security update that requires us to move much more quickly than we normally would, or for a fix where the risk of regression is overcome by the fix mitigating a problem that's already very impactful to

customers. These exceptions are very rare, in general our deployment tools and processes intentionally sacrifice velocity to maximize the chance for signals to build up and scenarios and workflows to be exercised at scale, thus creating the opportunity to discover issues at the smallest possible scale of impact.

Continuing improvements

Our safe deployment practices and deployment tooling continue to evolve with learnings from previous outages and maintenance events, and in line with our goal of detecting issues at a significantly smaller scale. For example, we have learned about the importance of continuing to enrich our health signals and about using machine learning to better correlate faults and detect anomalies. We also continue to improve the way in which we do pilots and flighting, so that we can cover more diversity of hardware with smaller risk. We continue to improve our ability to rollback changes automatically if they show potential signs of problems. We also continue to invest in platform features that reduce or eliminate the impact of changes generally.

With over a thousand new capabilities released in the last year, we know that the pace of change in Azure can feel overwhelming. As Mark mentioned, the agility and continual improvement of cloud services is one of the key value propositions of the cloud – change is a feature, not a bug. To learn about the latest releases, we encourage customers and partners to stay in the know at Azure.com/Updates (https://www.Azure.com/Updates). We endeavor to keep this as the single place to learn about recent and upcoming Azure product updates, including the roadmap of innovations we have in development. To understand the regions in which these different services are available, or when they will be available, you can also use our tool at Azure.com/ProductsbyRegion (https://www.Azure.com/ProductsbyRegion).

Cloud Strategy (/en-us/blog/topics/cloud-strategy/) Management (/en-us/blog/topics/management/) Updates (/en-us/blog/topics/updates/)
Advancing Reliability (/en-us/blog/tag/advancing-reliability/)

Subscribe (//azurecomcdn.azureedge.net/en-us/blog/feed/)

Explore

See where we're heading. Check out upcoming changes to Azure products

Azure updates (/en-us/updates/)

Let us know if you have any additional questions about Azure

Ask questions (https://aka.ms/azureqa)

Topics

Announcements (/en-us/blog/topics/announcements/) (2331)

API Management (/en-us/blog/topics/api-management/) (39)

Artificial Intelligence (/en-us/blog/topics/artificial-intelligence/) (248)

Azure Maps (/en-us/blog/topics/azure-maps/) (31)

Azure Marketplace (/en-us/blog/topics/azure-marketplace/) (149)

Azure Stream Analytics (/en-us/blog/topics/azure-stream-analytics/) (37)

Big Data (/en-us/blog/topics/big-data/) (655)

Blockchain (/en-us/blog/topics/blockchain/) (89)

Business Intelligence (/en-us/blog/topics/business-intelligence/) (119)

Cloud Strategy (/en-us/blog/topics/cloud-strategy/) (727)

Cognitive Services (/en-us/blog/topics/cognitive-services/) (126)

Data Science (/en-us/blog/topics/datascience/) (113)

Data Warehouse (/en-us/blog/topics/data-warehouse/) (222)

Database (/en-us/blog/topics/database/) (631)

Developer (/en-us/blog/topics/developer/) (1199)

DevOps (/en-us/blog/topics/devops/) (86)

Events (/en-us/blog/topics/events/) (257)

Government (/en-us/blog/topics/government/) (78)

Hybrid (/en-us/blog/topics/hybrid/) (86)

Identity & Access Management (/en-us/blog/topics/identity-access-management/) (89)

Internet of Things (/en-us/blog/topics/internet-of-things/) (386)

IT Pro (/en-us/blog/topics/it-pro/) (601)

Last week in Azure (/en-us/blog/topics/last-week-in-azure/) (92)

Machine Learning (/en-us/blog/topics/machine-learning/) (52)

Management (/en-us/blog/topics/management/) (391)

Media Services & CDN (/en-us/blog/topics/media-services/) (207)

Migration (/en-us/blog/topics/migration/) (41)

Mobile (/en-us/blog/topics/mobile/) (159)

Monitoring (/en-us/blog/topics/monitor/) (150)

Networking (/en-us/blog/topics/networking/) (238)

Partner (/en-us/blog/topics/partner/) (138)

Security (/en-us/blog/topics/security/) (410)

Serverless (/en-us/blog/topics/serverless/) (81)

Storage, Backup & Recovery (/en-us/blog/topics/storage-backup-and-recovery/) (702)

Supportability (/en-us/blog/topics/supportability/) (47)

Updates (/en-us/blog/topics/updates/) (582)

Virtual Machines (/en-us/blog/topics/virtual-machines/) (736)

Web (/en-us/blog/topics/web/) (373)

Articles by date

November 2021 (/en-us/blog/2021/11/)

October 2021 (/en-us/blog/2021/10/)

September 2021 (/en-us/blog/2021/09/)

August 2021 (/en-us/blog/2021/08/)

July 2021 (/en-us/blog/2021/07/)

June 2021 (/en-us/blog/2021/06/)

Full archive (/en-us/blog/archives/)