GETTING STARTED WITH CLOUD NATIVE APPLICATIONS

This is the first blog in our mini-series that illustrates how BMC was able to use agile development, cloud services, an Infrastructure as Code approach, and new deployment technology to deliver a new cloud native product.

Be sure to also read: 9 Steps for Building Pipelines for Continuous Delivery and Deployment, Infrastructure and How "Everything as Code" changes everything, and Five Best Practices for Building Security as Code into a Continuous Delivery Pipeline

Gartner defines BiModal IT as an organizational model that segments IT services into two categories based on application requirements, maturity and criticality. Mode 1 is predictable and traditional, with an emphasis on exploiting what is known. It's a reliable approach based on scalability, efficiency, safety, and accuracy. The challenge with Mode 1 apps is that the cycle times are long (i.e., months).

If you want agility and speed, then consider Mode 2 apps as part of your IT strategy. Gartner compares the Mode 1 style to that of a marathon runner. Mode 2, according to Gartner is like a sprinter, where work is exploratory and is often tested in short iterations, such as days or weeks.

Putting Mode 2 into Practice

Our team recently built a Mode 2 cloud native application on an Amazon Web Services (AWS) cloud and this project got us thinking about the unique enablers that led to creating a successful Mode 2 cloud app. Mode 2 apps are critical for meeting the requirements of digital business because they enable DevOps teams to accelerate the pace of innovation. I'd like to share with you some best practices in building these applications and getting your product to market quickly.

1. **Act like a start-up and focus on the minimum viable product (MVP)**

Startups usually have a solid vision and a focus on doing one thing very well. Software startups often focus on delivering the MVP quickly. This allows them to get feedback, iterate and develop better versions faster, rather than taking too long to develop what may or may not be the right product. We acted like a startup charged with solving a customer problem. In this case, the problem was trying to develop a product that could help customers simplify assessments and compliance reporting for cloud operations. Keep in mind that Mode 2 cloud native applications require what can seem like a maniacal focus on the customer problem, market validations, and delivering value to the customer by zeroing in on one or two use cases. I've seen many Mode 1 projects that took 2 to 6 months just to define the product to build! We needed to be more agile. We knew exactly what to build and had an uncompromising faith in the product and what was required.

2. **Use the Cloud for the higher-level services of managing the infrastructure**

Our team decided to go "all in" with AWS cloud to get the heavy lifting done – the infrastructure automation, routing, message buses, load balancers, server monitoring, running clusters and servers, patching them, maintaining them, and so on. This "all-in" approach helped get the product to market faster by eliminating the need for us to have to deal with some of the following challenges:

- Spending too much time on what seems like never-ending discussions related to determining the platform
- Ensuring portability to multiple clouds or multiple data centers
- Determining what should be on-premises or SaaS

3. **Use Microservices-based architecture**

We heavily used the microservices and 12-factor application principles in architecting our application to increase speed and agility. We designed six separate microservices based on key business use cases and functions.
Each microservice was independently and autonomously designed and built by one or two-person engineering teams. Each team had complete flexibility in not only picking the programming language, but also the data store. Of course, certain guardrails were clearly defined for all the microservices. For example, each microservice exposed a REST API with a declarative specification that was defined in Swagger and pushed to the AWS API Gateway. Each microservice chose its own database services if it needed persistent services. In addition, each microservice followed Infrastructure-as-Code practices, which I'll discuss in blog #3, where the full-stack definition was declaratively stored in AWS CloudFormation templates.

Empower your team
Our fully autonomous team was charged with making our own decisions and shaping the destiny of our apps. Doing endless comparisons and engaging in dialog that created paralysis by analysis were avoided at all costs. After six months of using AWS for a variety of applications, we continue to be delighted and amazed by the power of the platform. While AWS manages the infrastructure, our attention is focused on addressing business problems that matter and building apps to solve them. Today, all of our conversations in the team are about customer use cases, user stories and design patterns based on PaaS and server-less paradigms and DevOps. We don't need to spend time managing infrastructure such as machines, networking, firewalls, clusters of database machines or clusters for big data streams and so on. We let AWS take care of these while we focus on delivering value to the customer.

Stay tuned for our next blog on our AWS project to learn about how treating Infrastructure as Code helped drive quality and consistency in our development pipeline.