## MODULE 3: DEPLOY A PILOT APPLICATION IN IBM CODE ENGINE

#### Case Study 1 : Defining our Minimal Viable Product

**Introduction**

In this topic, we'll explore the concepts of creating a minimal viable product for our pilot application that we will then build and test in the lab.

Throughout this topic, we will attempt to answer the following questions:

**How are Minimal Viable Product(MVP) essential to the agile/garage methodology to enable leaner cloud development?**

**Recap**

In Module 2, Acme Airlines reached out to the IBM Garage team to help them get started with modernizing their operations. The Acme team then participated in several ideation exercises involving enterprise design thinking.

The team took a close look at the existing company culture and identified several areas of strength, as well as potential hurdles, along their modernization journey.

Using industry data on airline passengers, they conducted an empathy mapping exercise to better understand, on a personal level, who their target audience is.

The decision was made to begin Acme's digital transformation by transitioning its outdated legacy IT architecture to a more modern hybrid cloud infrastructure.

Using feedback from the To-be scenario, the development team will create a prototype mobile app and pilot it through a beta test program.

**Defining Your Minimal Viable Product**

Enterprise Design Thinking enables your team to gain a clear understanding of your users and empathize with them. Through workshops, you can learn how users do their work today and what they need, to do it better, faster, or more efficiently tomorrow. Based on this new knowledge, you define a minimum viable product (MVP) that addresses their needs.

An MVP is the least amount of design, development, and infrastructure that is needed to prove that your proposed solution meets the users' needs. After you define the MVP, implement just enough architecture so the team can implement the solution.

No matter the nuance, the intent of MVP is the same: reduce risk while you increase return on investment by building only what is necessary. "What is necessary" means the smallest possible version of a product that can be used to run a meaningful experiment to test key hypotheses and determine whether to continue investment.

An MVP is built with minimal investment and features but is focused on a viable solution to an opportunity. It's the foundation upon which to iterate to deliver measurable business outcomes. Each MVP is aligned to a business initiative, where you use a hypothesis-driven approach to achieve incremental business value. An MVP might have specific features that are used to gain insight into high-risk assumptions. It might not address enough key user needs for a scaled product launch.

MVPs are used for both learnings and for delivering rapid business outcomes. Many MVPs are production pilots to get the highest fidelity feedback. You must learn before you're ready to go to scaled production. The product owner works with the squad to evolve the MVP to have enough features and meet enough nonfunctional requirements to satisfy the target business objectives.

**How to Form the MVP**

Before you write an MVP, identify your assumptions and write hypotheses to test them. That information can help you form the MVP. What exactly is the smallest thing that Acme Airlines can build to test its hypothesis?

Using our scenario as an example, Acme Airlines wants to engage Alice, a millennial user. After the airline conducted user research and created an empathy map and a to-be scenario, Acme understands that Alice finds the entire travel experience to be stressful. Delays, long lines, and flight scheduling changes can be nerve-wracking.

Acme wants to make the whole process a more seamless experience. The airline believes that if Alice is happy and enjoys flying, she'll opt to fly more often and recommend Acme Airlines to her friends.

Using industry data and the ideas generated from the Enterprise Design Thinking sessions, the airline identified their assumptions and created a hypothesis:

"If we provide Alice, a millennial, with a fun application that gives her a hassle-free, predictable experience, that she can run at the touch of a button, Alice will book more flights. We will see that she internalized the value of airline travel through her continued use of the application."

**Hypothesis-driven testing**

What else might the airline need to know to test this hypothesis but reduce risk by not writing extraneous code? Alice is a millennial, so she will likely use her smartphone to book hotel accommodations. She also lives in the city, so it's likely that she will use features such as ride-sharing. Add that information: "A fun, responsive web application that gives her savings options that she can run at the touch of a button by using her smartphone."

Notice that the additions are in response to what the airline knows about Alice, not the result of business expectations or technical limitations. By staying focused on Alice, Acme Airlines can get the purest tests, which provide more reliable data.

Imagine that Alice lives is in Canada, where applications are often in both English and French. Code the initial MVP in one language, and then code it in another language after the first round of feedback is received. This approach saves time and effort by waiting until the essentials are validated. The MVP now states: "We will build Alice a fun, responsive web application in English that gives her hotel and car rental recommendations that she can reserve from a single mobile app."

The airline can ensure that its application for Alice achieves the results that it expects: to see Alice leave positive feedback and continue to use the application. The data from tests will either support that the business and design are going in the right direction or help Acme learn and shift. When Alice is happy and the application is providing her with value, she will continue using it.

By the end of the Enterprise Design Thinking workshop, your team is aligned on a clear MVP statement and can articulate the goals and future goals of the MVP. You're ready to develop the solution architecture for the MVP and move forward to inception as the first step to an MVP build.

#### Milestone 1: Build & Deploy a Pilot Cloud App

Using IBM Cloud CLI, we cloned the code-engine-microservices repository and built container images for the different microservices provided in the Dockerfiles.

We pushed those newly created container images onto Dockerhub. Those images were used to create Code Engine Applications.

#### Milestone 2: Generate Traffic

**Introduction**

In this topic, we will test the performance of our app using simulated traffic.

**How to test the MVP solution**

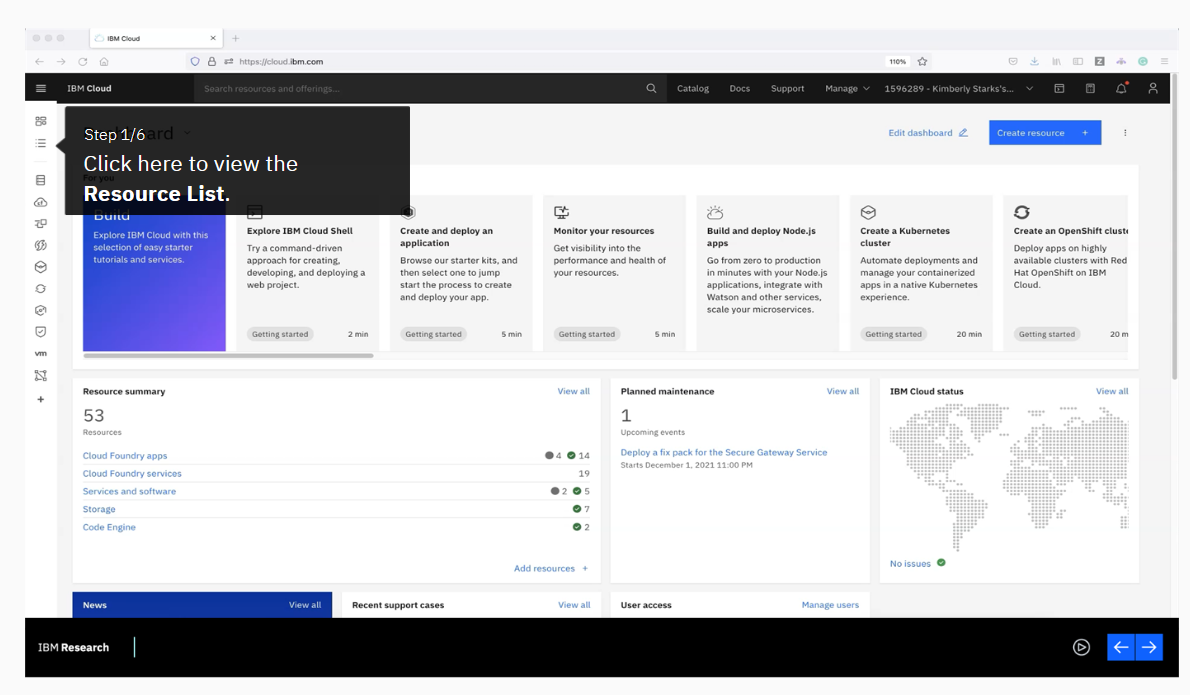
Some hypotheses can be tested without any coding, and if that’s the right MVP, of course, we do that. But the Garage has a bias toward building production pilots — we believe the best way to learn is by putting something real in the hands of real users.

Figuring out how to get something valuable into the user’s hands in, typically, six to eight weeks requires as much creative thinking as identifying the big idea. This is why the IBM Garage views Enterprise Design Thinking and Lean Startup as two parts of a single method, not two separate phases of a project.

In this part of the Lab, we are going to test what we have just done by simulating what it would be like to have many users accessing our "Bee Travels" project. This will allow us to watch the number of running instances of the application auto-scale based on the changing amount of traffic.

**Generate traffic**

Since Code Engine is a fully managed, serverless platform, the number of instances running for each application will auto-scale depending on the maximum number of concurrent requests per instance of incoming traffic to each application. In this part of the code pattern, we are going to generate traffic to the UI application of Bee Travels and watch the number of running instances of the application auto-scale based on the changing traffic.



**Congratulations**

Without having any knowledge or interaction with the underlying infrastructure of Code Engine, you have successfully completed the following:

* Built container images for Node.js and Python microservices
* Created/Deployed a workload to Code Engine consisting of public and private microservices
* Secured an external application
* Independent auto-scaling on a per-microservice basis

All of this was completed by only specifying desired runtime semantics (ex. whether to scale or not) and Code Engine took care of the rest.