serverless.com

6 Things to Know Before Migrating An Existing Service to Serverless

Marcia VillalbaAug 7, 2017

7-8 minutes

Last year, my company decided to make the plunge. We were going to go Serverless! Except...most of the resources about serverless architectures are about how to start from scratch, not how to migrate existing services over.

We spent eight months figuring it out along the way and, for all of you serverless hopefuls, we made a cheat sheet. These are the steps that worked for us:

1. Identify the problems
2. Train the existing team
3. Create a Proof of Concept to verify that the problem is solved
4. Optimize the solution to take advantage of the cloud
5. Automate your Continuous Integration/Continuous Deployment pipeline
6. Automate your testing

1. Identify the problems

What problems will serverless solve that your current solution does not? We (for example) wanted serverless to help us (1) lower operational costs, and (2) give us an easier way to replace a bunch

of legacy systems with a small team.

At my company, we decided that since AWS Lambda can be used as a glue between different AWS managed services, it was the best option for us.

2. Train the team

After you have defined the problems, it is easier to define (more or less) what set of technologies can help you. Your developers are the ones who already know your business logic and service requirements better than anyone else. So instead of heavily outsourcing, I'd recommend that if no one on your existing team is familiar with serverless, it is time to train them.

We encouraged our developers to go to meetups, conferences and to spend time (even work hours) tinkering with new technologies. In addition to that, we did hire a Serverless consultant to show us how to think in an event-driven manner and make sure we were following best practices.

In my opinion, one of the reasons our project was so successful was the weekly workshop meetings we held. They lasted 3-4 hours per week, and we used that time for discussing new serverless-related topics, solving issues and implementing solutions. We all got to learn a lot from each other, and workshop was very inspiring.

3. Create a Proof of Concept and validate your hypothesis

After you have identified the key problems to solve, and after your team has a better understanding of the tools available, it's time to create Proof of Concept. By now, your team should have an idea of what to do next—a "hypothesis". A Proof of Concept will help the team validate their hypothesis.

Remember: a Proof of Concept is *not* production code. Use it to

focus on solving your problem and get rid of it. Your Proof of Concept should go to the trash after validating your hypothesis.

In our project, we developed five Proofs of Concept. Recall that our problems were: (1) to replace legacy systems and (2) reduce operational costs. Our hypotheses were as follows:

To replace legacy systems:

- AWS Cognito will replace the existing authentication system
- DynamoDB will replace the existing Riak database as our NoSQL database
- AWS Lambda + S3 + Elastic Transcoding will replace our existing transcoding process
- AWS Lambda + API Gateway + S3 will replace our existing image resizing and provide better caching mechanisms

To reduce high operational costs:

- A workflow using Step functions + S3 + AWS Lambdas will replace our existing EC2 instances

By implementing these Proofs of Concept, my team ended up validating all of our hypotheses.

4. Optimize the solution to take advantage of the cloud

This step is critically in line with the previous step. When thinking about your Proofs of Concept and new architecture, it's important to take full advantage of the cloud. As in: don't just grab your instances and decompose them into AWS Lambdas and API Gateways using a DynamoBD; try to think about how to take advantage of cloud-managed services, like queues and caches.

Also remember that by migrating everything to Serverless, you are transforming the architecture of your system into an event-driven

architecture. In an event-driven architecture, events move around your system and all the services are decoupled from each other. AWS has a lot of services that can be used to manage events communication, like queues and streams. S3 is a great place to store your events. DynamoDB Streams can be used to let other services know that there was a change in your DynamoDB database.

5. Automate your Continuous Integration/Continuous Deployment pipeline

Since people can disagree on what exactly a 'microservice' is, I call a microservice a collection of AWS Lambdas and other resources related to some very specific domain, like authentication, application management or transcoding. Serverless architectures involve lots of resources being deployed into different environments in this way. When you have several moving parts, one way to make things simpler is to automate everything you can.

On my team, we used Serverless Framework to organize our projects and to automate microservice deployment into different environments. We wanted to define all our infrastructure configuration as code. The Serverless Framework helped us to do that, as all the resources needed in the microservice can be defined using Cloud Formation notation in the Serverless YAML file.

We used Jenkins Continuous Integration server to take care of running the Serverless Framework deployment in three different environments (development, staging, and production). For each environment, we used a different AWS account. We wanted to have three accounts so we could take advantage of managed resources soft limits as much as possible, and also to keep the different environments isolated.

6. Automate your testing

Testing is undervalued by a lot of teams. In a Serverless and event-driven architecture, the complexity of the code moves to the architecture itself. Because of this, testing at all levels is critical for peace of mind. Test, test, test. Also, test.

Unit tests will help you to make sure that your business logic works. For your managed services, you should write integration tests against them. It's important to define clear interfaces between your services and the managed services so that you can have control of what is going in and out of your service.

Don’t forget about End-to-end Testing. In an event-driven architecture, the services are decoupled from each other; as a result, it can be hard to know how the events are moving around the architecture.

Conclusion

Migrating an existing service to Serverless was work, true-but honestly, it was also fun. The two most important things to have in mind when migrating one (or all) of your existing services to serverless are:

- optimize your architecture to take advantage of the cloud
- remember that your new architecture is an event-driven one

Best of luck, and see you on the other side.