


## CS 470 Final Reflection

### [CS 470 Cloud Development Presentation - Ben Verrill](#)



# The Serverless Cloud



## Database

- MongoDB and DynamoDB are both NoSQL databases that support key-value pairs
  - MongoDB has greater deployment options and less complexity for queries
  - DynamoDB is AWS specific but is faster to get started with and easily integrates with other AWS tools

We used DynamoDB to create, read, update, and delete records within Question and Answer tables using REST methods.

A workflow for inserting an Answer record might look like the following:

- An API call is routed to a specific Lambda function containing a POST request with data
- Lambda sends a JSON object to DynamoDB containing HTTP POST method, data to be inserted, and what table to insert it to
- DynamoDB parses JSON request and carries out action and then sends response code back to Lambda function



Benjamin Verrill

[Benjamin.verrill@snhu.edu](mailto:Benjamin.verrill@snhu.edu)

Southern New Hampshire University

December 12, 2022

## Experiences and Strengths

This class has helped me from a professional standpoint in a few different ways. First, as this is my final course in my pursuit of achieving a computer science degree, it has helped to round out my development experience for how applications are actually deployed. Prior, I've built many different types of applications, but they were always simply deployed locally, and I shared the code in GitHub with whoever needed it. With this course, I've learned how to containerize apps and distribute them to cloud environments. I think this alone helps me become a stronger candidate for development opportunities because in this business it is expected that individuals have a broad knowledge of all aspects of development. DevOps has become an extremely popular practice, and having people coming into a business that have an understanding of the ways software is developed and released means less time onboarding them and faster time to bring the organization value.

Another way this course has helped me is with my current role. I'm an account executive for an engineering company that provides consultation and professional services for Datadog. In this business, having an understanding of DevOps allows us as salespeople to be far more knowledgeable about the everyday pains our customers are going through and gives us credibility when advising them on how to strengthen their observability practices. The different areas that we touched on in this course are some of the exact things my engineers do for our customers. I was recently talking to one of my colleagues about how awesome it was that, just a couple of months ago, we created a statement of work for a customer that included work with Lambda in AWS. I had no idea what was involved with that (thankfully my engineers do), but after taking this course I've now worked in Lambda directly and can now speak more articulately about the value my company can bring to others in this regard. Now, at the end of my degree, I feel like I have a strong grasp on development and deployment and, with my soft skills from

years of sales experience, I have a strong background that would allow me to make a career change should I choose to.

## **Planning for Growth**

We've learned a lot in this course. We learned how to containerize applications, deploy them to cloud environments, create APIs in a centralized environment, write and use serverless code, and secure access using Identity Access Management tools. The beauty of containerization is that we can continue to break down our app into various microservices to better scale the app. We can add new features to our app by placing those new features in a different container that speaks to the others, and should certain functions see more traffic than others, we can simply spin up more containers to meet the current need for resources so that our customers always receive the best possible experience. We can automate the configuration and deployment of our containers using configuration management tools. We can orchestrate our many containers using tools like Kubernetes or Docker Compose, which handle containers at scale. Using cloud environments like Amazon Web Services, Google Cloud Provider, or Microsoft Azure, we can easily scale our app as it grows so that we only need to pay for the resources that we're using and cut down on management costs.

One thing we didn't talk about much in this class is the use of monitoring tools for troubleshooting and error handling. As we scale our app and find ourselves working in various cloud environments with different tools and lots of containers, it can be really hard to narrow down where things are going wrong. Introducing monitoring and setting up alerts for when certain metrics, like latency, go over set thresholds can help us figure out when something is going wrong in our app and where it's happening. Of course, this could be an entire class in itself as there are many things to observe and alert on, but it's a key part of maintaining a deployed app.

Cost is another thing we didn't dive too deeply into. How do we understand the future costs our app will incur? This is a tough one to get right, but thankfully most cloud providers have built in tools to track and predict future spending. It helps to understand where your costs are coming from so that as you utilize more tools and resources, you'll know what will drive costs up faster. For example, spinning up containers that you control may be easier to understand future costs as you control and maintain them on your own, whereas serverless tools like Lambda may dynamically increase in use at any given time making this type of forecasting much harder. I think monitoring, again, comes into play here as well. To know when to scale your app, you need to know how much of your resources you're using at a given time. In cloud environments, we can easily increase or decrease the resources that we're using which makes tracking saturation a bit harder, but if we're getting constant alerts that we're seeing spikes in traffic or over utilization of resources for things on prem or beyond a baseline we've set in cloud environments, it may be time to start thinking about scaling our app. Setting baselines is also tricky, because we want them to be low enough to be caught so that we can effectively increase the resources needed before they effect the user experience, but not low enough that we start paying for resources that we don't yet need. The cost tracking tools that cloud providers offer also have forecasting functionality that can help here. Like with anything else, there's a balancing act between how much we want to spend and how much risk we're willing to expose our application to for poor performance when it comes to scaling it.