{Christina:

Hello everyone. My name is Christina, and I am joined by my teammates Michael, Sarah, and Travis. Today we will be presenting on one of the many ways teams can host content on Amazon Web Service products to be consumed by end users.

(title page slide. Need a title for this tech talk)

Our full-stack application relies on numerous systems we’ll be walking you through the configuration for, but a quick snap-shot of those systems are as follows. A front-end React application sits inside an S3 bucket, serviced by CloudWatch and Route 53. This is paired to a Python Flask App and Node.js Express App that sits inside an EC2 instance, also serviced by Route 53. This gives us a very secure application from end-to-end and will be the basis for our how-to, today. (maybe a slide that’s this workflow??)

First, we’ll start with Route 53, Amazon’s DNS web service. After you log into your AWS Console account and navigate to the Route 53 page by using the search bar in the center of your screen (route53\_16). Here you’ll navigate to the Register domain section of the page (see Route53\_5) and after entering your desired domain name, you’ll check it’s availability. (see Route53\_2) Assuming the domain is available, you click add to cart and proceed to the next page where you enter the appropriate information into the fields (see route53\_3) as pictured. After this you click next on the bottom of your screen and are taken to a confirmation page (see route53\_4) where you select the appropriate options for renewal, read and agree to the terms and conditions, then verify your e-mail address before clicking complete order. You’ll receive a couple of e-mails confirming your new domain and informing you when registration is complete. Upon receipt of these e-mails, you’ll want to navigate back to Route 53, and click the icon in the top left of your screen (see route 53\_6) to expand the navigation toolbar. (see route53\_7)

Clicking on Hosted Zone will take you to a list of your current Route 53 resources, including your new domain! (route53\_8) Now you’ll click on the hosted zone you just created and navigate to DNSSEC signing (route53\_9), an important step for validating that DNS responses haven’t been tampered with. Here, you’ll fill in the appropriate information (route53\_10) and select Create Customer Managed Key, allowing Amazon to generate the appropriate records and handle key management for you. After your transaction has gone through, you’ll need to navigate back to the hosted zone, into the DNSSEC signing tab. You’re now looking for the button, “View information to create DS record.” (route53\_13) Open this tab and expand the Establish a chain of trust tab, then the Route 53 registrar. (route53\_14) Take note of these values, you’ll need them shortly Now you’ll want to navigate back to the sidebar and click on Registered Domains (route53\_11). After selecting your domain you’ll look for the DNSSEC status on the right-hand side (route53\_12) where you want to click Manage Keys. Now you’ll take the information from the Establish a DS record tab and fill them into the fields in the pop up (route53\_15). Once that’s done, click add and wait for the updates to propagate! Now that we’ve ensured our new domain is configured to correctly sign DNS responses with it’s public key, let’s make sure there’s actually content for a user to load. So here I’ll hand things off to Sarah to discuss how a Route 53 domain can be paired with an S3 bucket, and a CloudFront Distribution. (Idk maybe the work-flow slide from before, but like there’s an arrow tracking where we are in the presentation? Where like the first slide it goes {Route53} -> {S3} -> {CloudFront} -> {EC2}->{Demo} and like, the arrow in the first slide points to Route53, but now it’s updated to CloudFront?

}

{Sarah:

Thank you Christina, and hello everyone, I’m Sarah and I’ll be walking you through the rest of the front-end configuration we can use, starting with S3, a cloud storage option offered by AWSx`!

First we’ll navigate to AWS’s S3 (S3\_1) page were we’ll click on Create Bucket.(s3\_2) We then need to specify a unique name for our bucket that must be the same as our URL registered with Route53, and fill in the rest of the information, making sure to uncheck (s3\_3) “Block all public access” Then check the acknowledgement regarding public access (s3\_4) and click create bucket at the bottom of your screen. Next, navigate back to the S3 console and select your new bucket from the list. You need to navigate to the properties tab (s3\_5) and navigate to the static website hosting. Click Edit and enable static website hosting. (s3\_6). This brings up a dropdown that you need to fill out. Index document will be index.html, and error document will also be index.html for a React app. This may be different for other application types. Click save changes and if not automatically redirected, head back to the properties tab, scroll to the bottom and look for your new Bucket website endpoint. (s3\_7). Take note of it, this will be important later. Now that we’ve got a bucket configured correctly, we need to actually upload an index.html for the bucket to process. This requires us to build our React program. (cmd\_1). Navigate to the appropriate folder your react project is located in, and run the command npm run build. After your project is finished building (cmd\_2) you’re ready to upload it to your new S3 Bucket!. Navigate back to S3 and click on Upload (s3\_8) and click on Add Files (s3\_9). Now navigate to your React project in the file browser, until you find the build folder (s3\_10). You must select the entire contents of this folder (s3\_11) which may be slightly different from what’s pictured here, and click Open. Now click Add Folder (s3\_12) and select any folders in the build folder (s3\_13) and click open. Now that all the files are selected, click upload in the bottom right of your screen and await the results (s3\_14). When the upload is finished, click close. Navigate back to the Objects tab of your S3 bucket, and select your new index.html, then click on the actions dropdown, looking for the Make public action. (s3\_15). Click make public, wait for it to update, then click close in the top right. (s3\_16). One final step to ensuring users can gain access to your content, so head over to permissions and scroll down to Bucket policy. (s3\_17) . Click edit and construct the following JSON policy for your bucket (s3\_18). Naturally, you’ll replace the word uniquename1123124 under the resource section with the name of your own bucket. Click save changes at the bottom and you’ll notice that your bucket has a new tag, (s3\_19) which identifies it as being publicly accessible. Congratulations, you’re one step closer to have a fully hosted application! If you take the time to navigate to that bucket website endpoint (s3\_7) you’ll see that your site is already hosted at that endpoint! (site\_1), but now it’s time to get it all connected to your Route 53 domain name.

Navigate to route 53 (route53\_16) and choose hosted zones (route53\_7). Find the Create Record button (route53\_17) and click on switch to wizard (route53\_18). Now ensure simple routing is selected and click next (route53\_19). Choose “Define Simple Record”. In the popup (route53\_20), leave the record name and type to their defaults, and under Choose Endpoint (route53\_20 select alias to s3 website endpoint. Then select the region you made your bucket in, and in the enter s3 endpoint it should dropdown with the bucket you created earlier (route53\_21). Make sure Evaluate target health is set to no and click define simple record. Just like that, if you navigate to your Route 53 domain you should successfully navigate to your website! Congratulations, you have a fully hosted react application! Thank you everyone, I’ll now pass things off to Michael to explain how our back-end hosting configuration can be achieved!

(idk, I feel like this just kinda… trails off tbh).

}

{Michael:

Thanks Sarah! We now have a frontend application which can serve up interactive webpages to our users, but we still need backend infrastructure for any processing and storage of data that needs to be done. Amazon provides us with a few tools which let us achieve that, the first of which we’ve already talked about. Amazon S3 provides simple and scalable object storage, which can be useful for holding data generated by backend operations, or to store user files. S3 offers multiple storage classes for different use cases, such as Infrequent Access, which focuses on data which is accessed infrequently, but still requires rapid access when it is. Another example is S3 Glacier, which focuses solely on long-term archive storage, allowing you to give up access speed in return for cheaper storage.

That lets us store general data, but for data we want restrictions on to enforce consistency, such as user data, we’re going to need a database, and Amazon RDS allows us to set that up. Navigate to Amazon RDS and then to Databases on the sidebar. Then, click create database, and create a database of your choice, making sure to read customization options and the estimated cost summary, because there are several tiers available. Open the newly created database and note the endpoint under the connectivity and security tab. This is where you’ll direct requests to access your database.

Ok, so, the combination of Amazon S3 and RDS already gives us plenty of options for storage of backend data, but we still don’t have any way to execute code which modifies or extracts information from that data. Of course, AWS provides solutions for this problem as well. The ideal solution for your problem is going to depend on the manner of the task being done, but the two main solutions offered I’m going to talk about are Amazon EC2, and Amazon Lambda.

Amazon EC2 provides you with a persistent virtual environment which can handle consistent high workloads, while Amazon Lambda provides event-driven code execution. EC2 requires more setup and maintenance, but if your application requires any kind of constant processing, this is a great place to look. Lambda, on the other hand, is very lightweight, and only executes the code you want when a specific event happens, and is a get place to get started off.

The beauty of Lambda is its flexibility, and here’s just a simple case for you to try with your newly set up services. Navigate to Lambda and create a new Lambda function, and change the code to make this function do something simple such as printing Hello World to the console. Now, add a trigger to cause this function to execute on a file put to that S3 bucket we created earlier, and then go add a file to that bucket. If you come back to your Lambda function, you should be able to find a console log under the Monitor tab, printing that Hello World message in response to the addition of the file. This is just a very basic example of Lambda, but you can change both your code and triggers to suit your needs. For example: You have some frontend code that when the user interacts with, sends off a request to API Gateway, Amazon’s hub for API requests, where Lambda would have a trigger, with S3 storage or a database as an endpoint for its data. Linking all these things we’ve shown you together allows you endless ways to build out your application from here to suit your needs, and now Travis is going to show you a quick demo of getting the frontend environment set up so you can get working on it yourself.

}

{Travis:

}