

**Characteristics of distributed systems**

we're focusing specifically on web applications.

There's going to be a lot of different architecture styles, so you can still have a client-server sort of architecture where you've got a client talking to a server for info and presenting it back to the user and clients sending data back to the data for persistence, things like that.

You can get to these three tiers or end tier architectures where the client logic moves to a middle layer, so clients are stateless. End tier, you may have again additional layers if there are things like caching or other app services.

You can have a cluster style, where clusters of machines work together.

A peer-to-peer network where there's no single controller, but all these machines have some sort of equal responsibility and you can do a space based sort of thing where you're using SOA, and REST, of an adventure of an architecture, where you have this distributed machines that all scale independently, and have shared nothing. So a lot of different architecture styles.

-🡪**Data sharing** is going to be different. You might have a :  
**shared databases** where everyone's dependent on a single data source.

**Synchronization** is going to come into play if you're having different replicas of data of some sort of no single master shared nothing, but where data has to be synchronized among multiple nodes and so that's where you're going to look at **replication** of **messaging**, or other different technologies and ways to keep things in sync.

**Caching** is going to come into play as you have data sharing, is how can I cache information that might be slow changing to make sure performance is strong when I am expanding geographies and dealing with latency.

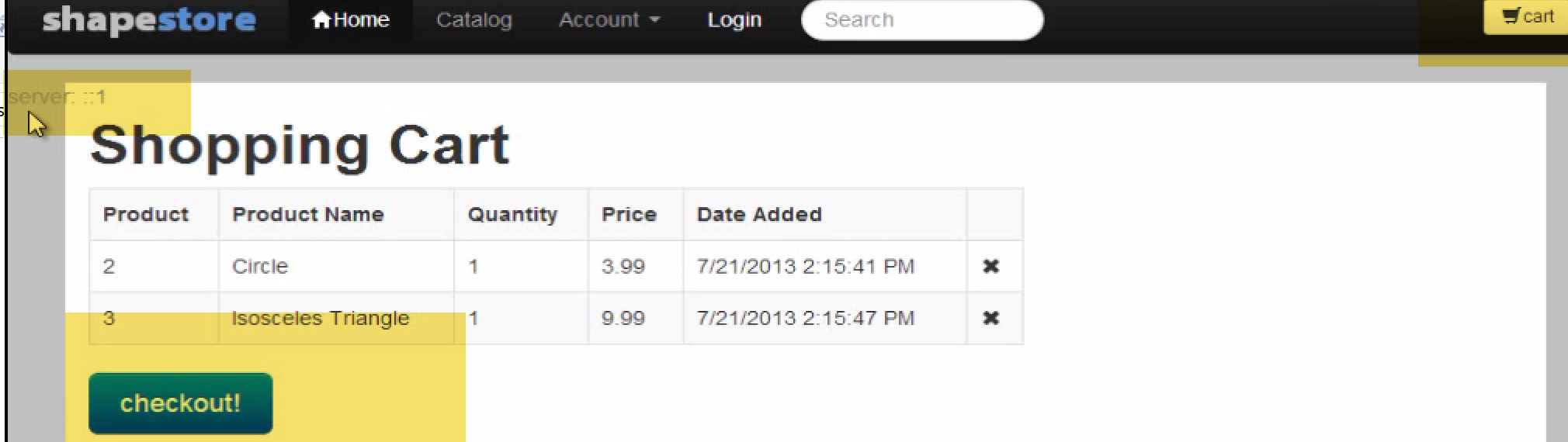
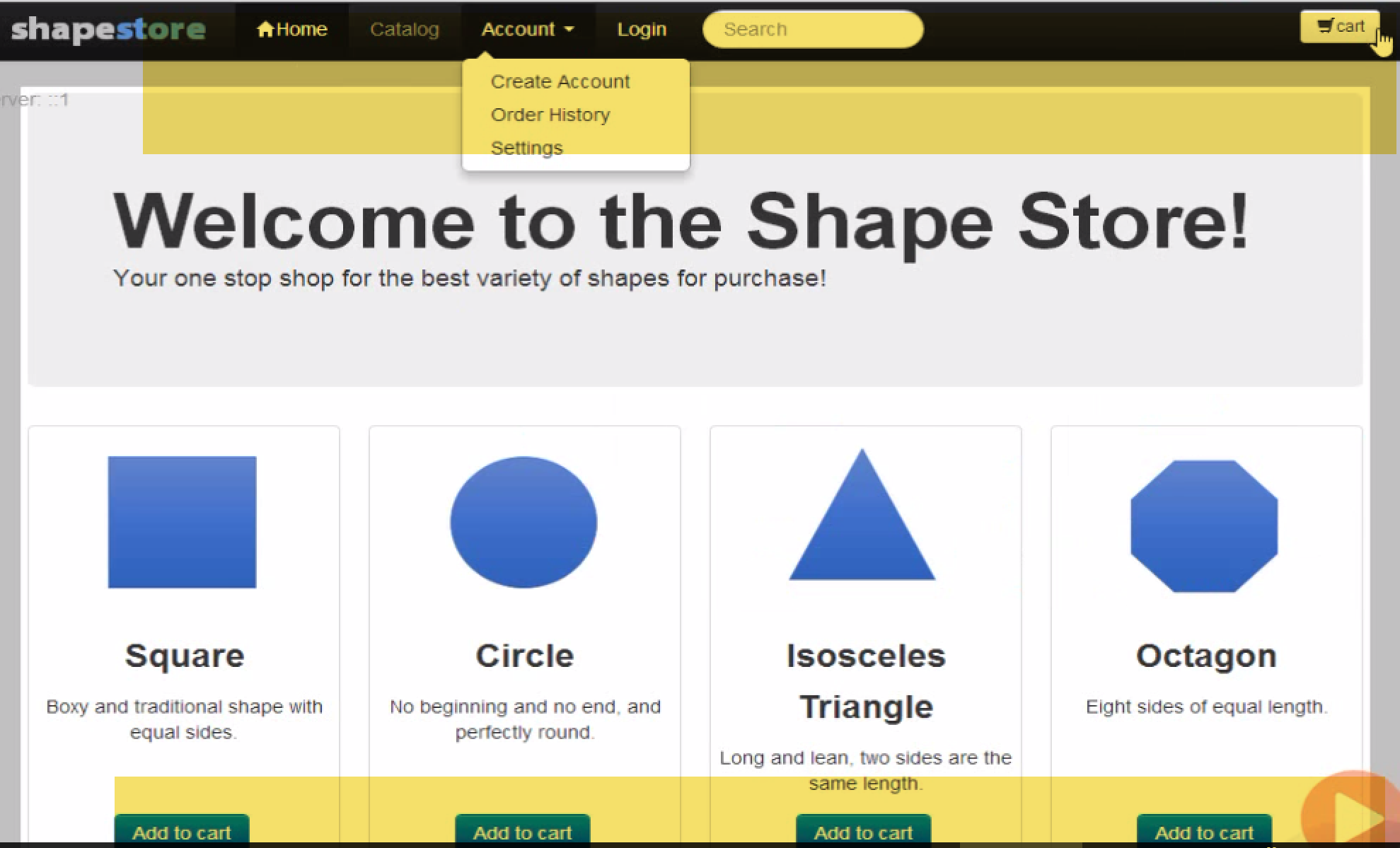
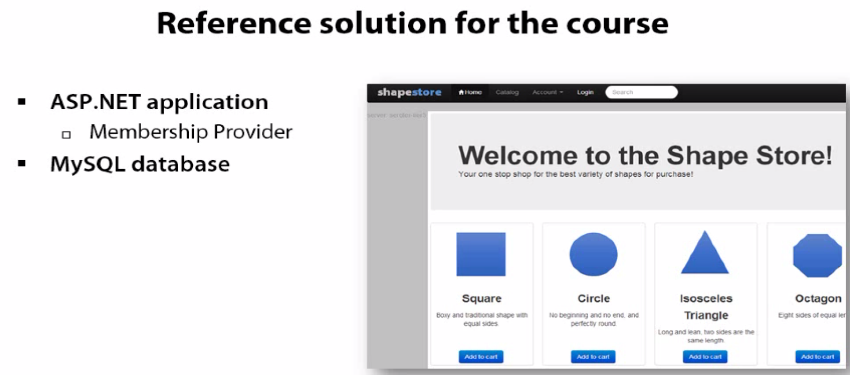
A lot of different **fault tolerant techniques** when you're dealing with distributed web systems. Whether you're working with timeouts and retires to bake in, to make sure you don't have worry about if a single thing fails, can you try it again. Having isolation, things like separate threads, so that again a single failure doesn't cascade and that's where you get into things like circuit breakers, where how do I make sure that I don't have cascading errors and if something fails, I can even switch over to a stub or something else that might be using caches, or no responses, or even default data to handle a failure of a data subsystem or even just use eventual consistency and drop a message into a queue and it'll synchronize later. I'm going to use fail fast techniques in these distributed systems where again, latency is often worse than something failing completely. So you want things if they're not performing, fail and move on and there's going to be a lot of these

core principles like **availability**. That's going to be something you're going to focus on a distributed web system you have to architecture for. You can't assume it's baked in on any platform especially the Cloud. **Reliability**, again this thing has to be consistently working when you're distributing over this, you have to think about that in your architecture.

How can you **scale**? That means how can I continue to increase and decrease load and have the environment grow and shrink as necessary without reducing my functionality?

**Performance**, can I get consistent performance? Can I handle spikes in performance? Can I make sure that regardless of what I'm sending at the system, it's going to perform okay?

And then **manageability**. How can I make sure that even if I'm expanding all of these different layers and subsystems and owners? I'm not dealing with a management nightmare where I can't ever trace an actual problem….

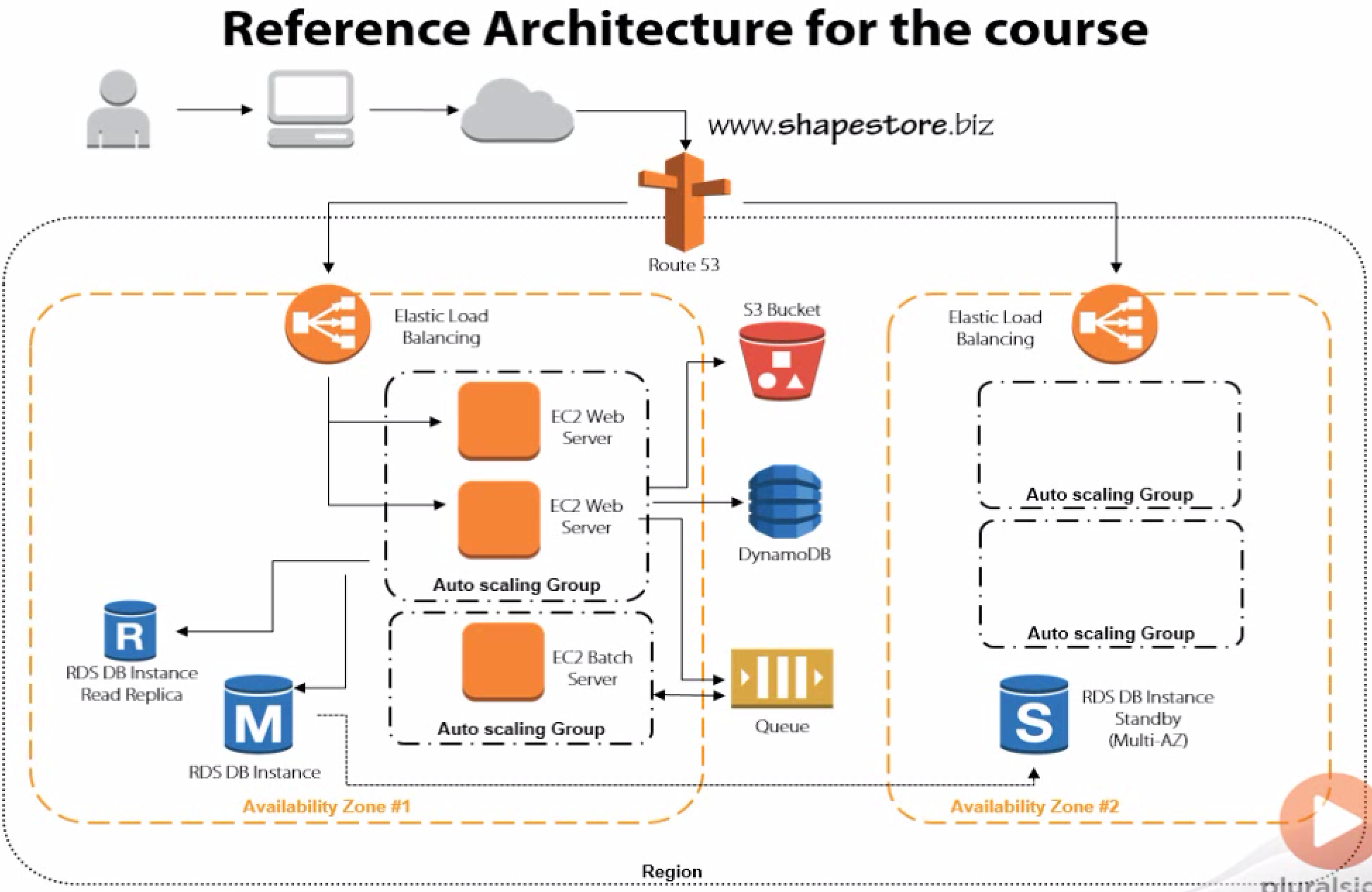


**Reference architecture for this course**

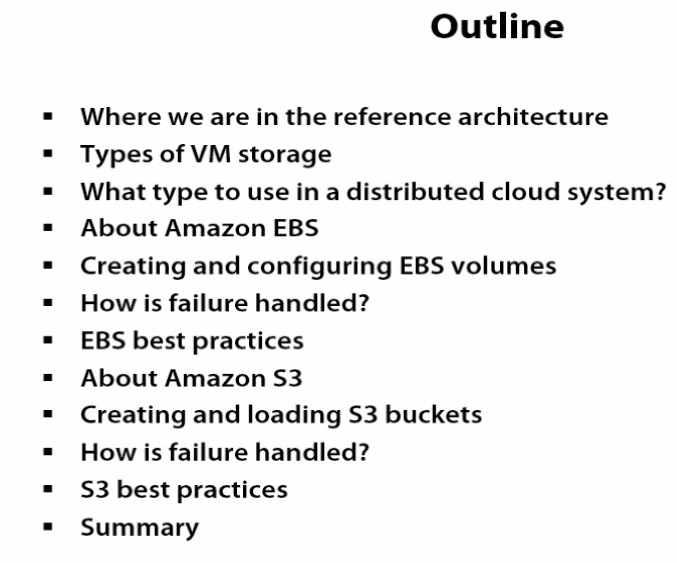
Web traffic that comes into shapestore.bix, I went ahead and registered that domain. That data is going to come into an Elastic Load Balancer, which is going to route traffic to one of many different servers, those are all part of a scaling group so they'll scale as needed.

Static Content is stored in Amazon S3,

Session State being stored in DynamoDB, so it can expand servers.

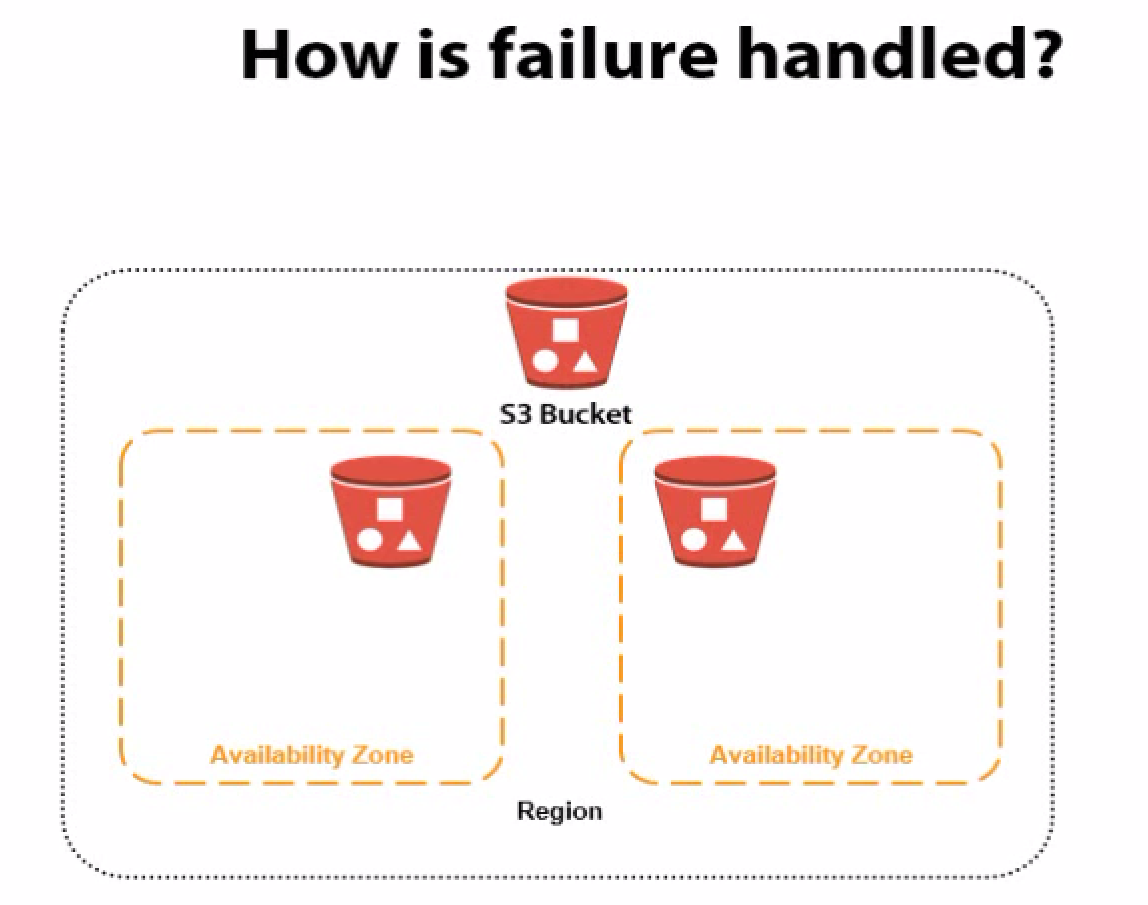
I'm storing data in RDS, the Relational Database Store for things like the product catalog and I'm getting the orders in there. But of course, orders are first going into a queue and I have a separate set of servers that are going to read the orders from the queue and process them, so that if I have a huge spike, I'm not putting a constraint on the database directly, I'm just going to keep filling up the queue and I'm going to have servers that process those kind of asynchronously, so I reduce my coupling. That relational database is automatically replicated to a different availability zone, so I have a standby server. So I could lose an entire availability zone and everything would keep running.

**Provisioning Durable Storage with EBS and S3:**

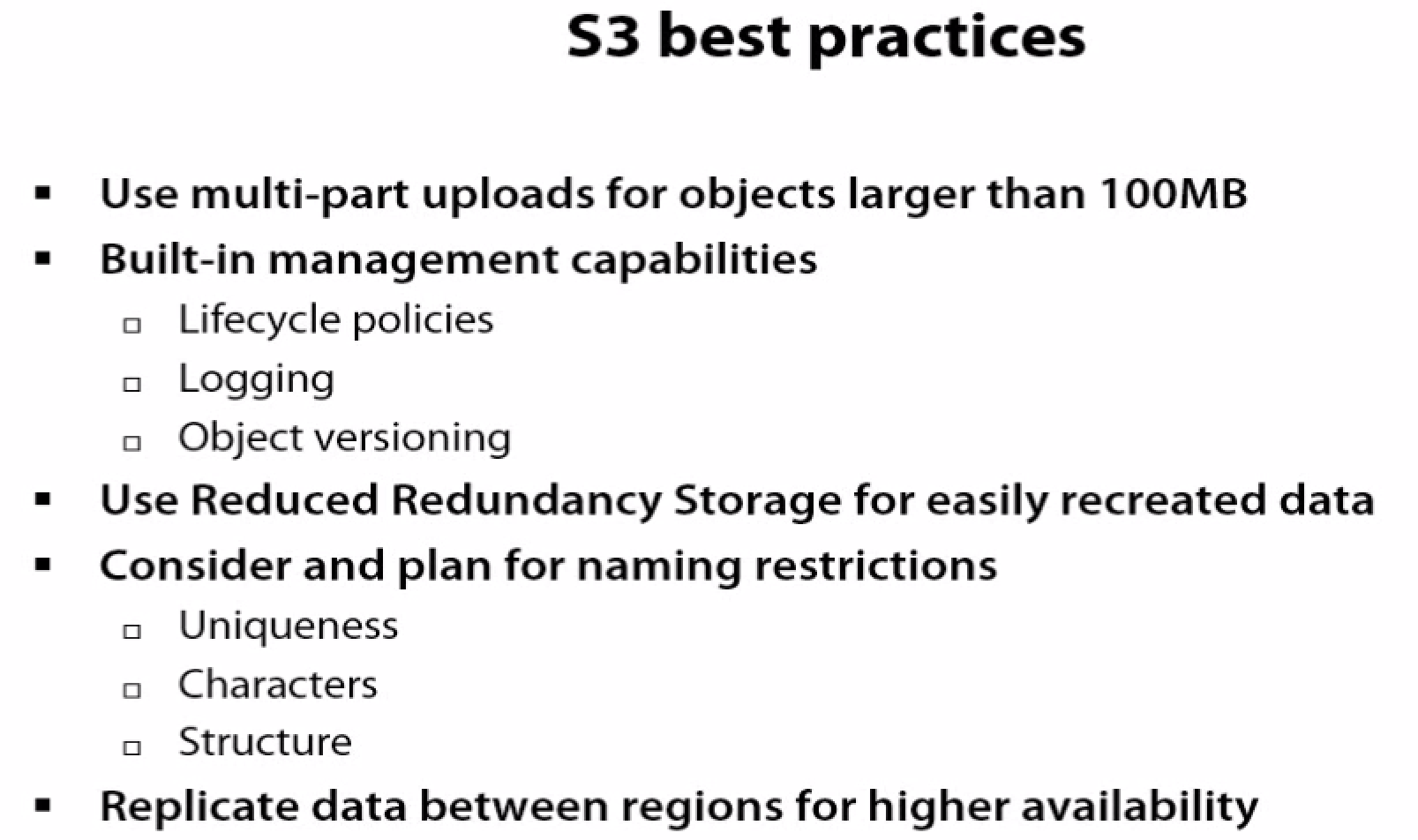
****

**How is failure handled?**

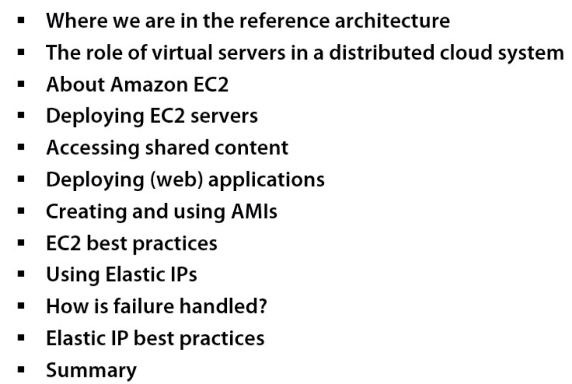
Handling failure with S3 is pretty straightforward mainly because everything's geo redundant by default. So even if I lose an entire Availability Zone, everything really just keeps working because you've got that redundancy, the address doesn't change, so when I'm accessing that bucket URL that data's while spread across AZs I'm really funneling it through one address. So even if I lose an Availability Zone, nothing really bad happens and as you saw as well, another form of failure is having a poor performing website, I can also use S3 to host a static HTML website if I want to point to a, let's say a less busy static highly available site, in the case of a degraded performance.

****

**S3 best practices**



**Adding EC2 Virtual Machines**

****

**Where we are in the reference architecture**

So far in our reference architecture, we've added object storage through S3, we've added DynamoDB for session data, we've added relational databases for some of our master data, and we've added queues to server as that proxy between the frontend web application and then the actual order processing pipeline. In this module we're going to jump in and we're going to add now the actual EC2 servers that act as webservers, as well as the batch processing server that pulls data from the queue and talks to the database.

The role of virtual machines in a distributed cloud system

So virtual servers as you would imagine play a pretty critical role in your web applications or any distributed system. This is where you're actually doing hosting of web applications for example. Or you're providing data or business services, again some of the logic of your actual application needs to be running somewhere and in this case they may take the form of a data tier, a business services tier in your application. Data processing batch processing kind of worker roles, headless servers that don't have a web application component, they're simply chewing on data or processing information. And then potentially running your own database. We talked about that in the database module where you may use a virtual server to install your own version of database software that you want to manage and maintain and that's a perfectly valid use of it in a cloud system. It's important though to treat virtual servers as disposable resources, that's one of the biggest mind shifts you probably have to get into when working in the cloud. Not treating servers as something that has to be provisioned to capacity upfront, needs to make sure it's constantly online, it never fails, that's not the way it really works in cloud systems. Instead, treat a server as a unit of scale only and really treat the templates as the units of scale. So I'm able to add servers, it's very elastic, I'm adding them, I'm removing them, that should not cause any change in the behavior of my system itself. It may change performance as I add more, I could do more things, but the behavior of the system shouldn't change as I scale up and down. So treat servers as something that might run for a temporary period of time and that's going to encourage you to design very stateless systems that don't have to have a lot of persistent information on the VM itself. If it happens to tip over there's no catastrophic impact because you've architected properly.

About Amazon EC2

Amazon EC2 is probably the best known of the Amazon services, it was one of the first after S3 and it's, you know, what its claim to fame is if you will it's being able to provision virtual servers on demand in the cloud. Launching them quickly, scaling them up and down, pay as you go, that's really the value of cloud computing in many cases is being able to provision machines this quickly. You do have root access, so this isn't an abstract layer where like a platform is a service where you can actually access the machine itself. In this case you're an admin on the server, you can access it just like you would any other server, except you don't have physical access, but you have access of the operating system at the root level. It is deployed globally, so EC2s available in all the Amazon datacenters. Meaning you can stand up servers in Singapore or Oregon in the US, South America, wherever you'd like. Deploy servers with a click of button without doing anything different. There's a lot of different instance sizes, so you don't just provision how much RAM and CPU and storage you want, you pick from some pre-canned instance sizes like small or micro or extra-large or triple extra-large that are going to meet different needs and be a good fit for different types of workloads. Instances can fail or terminate for reasons outside your control, so there could be an underlying hardware failure, there could be things that do cause an instance to fail. So again treating servers as disposable and architecting for that is going to be a good thing to be aware of upfront. EC2 instances are launched from AMIs, which are Amazon machine images. These are templates that have an OS, application software on them, and they can be provided by Amazon themselves, third parties, developers, and you can create your own. These are a nice way to create templates of gold versions of software and things like that, that you don't want to have to worry about. Instead of recreating this from scratch every time, I can use a template that has the software I typically use on it or at least the way I like my machine set up. So again this is just kind of the library of templates that you can maintain yourself or just use ones created by others. And AMI is going to either be backed by EBS or instance storage and we discussed this a little bit in the storage module. Where you have to again remember that the root device may be on an EBS volume if it's over 10GB, which is going to be most Windows instances. And that's going to be durable storage, which means I can shut down the server, terminate the machine, and technically not lose any of my data. If the root device is on instance storage, which is a default for a lot of Linux machines, if I stop the box or if I terminate the box I've lost the data. It's ephemeral storage. So, both of those are valid versions of AMIs, you just got to notice that when you look at the catalog. Access to these servers from a network perspective is controlled by Security Groups. So these are kind of similar to inbound firewall policies where you're picking which ports and which source IPs and protocols are allowed into the server. So for a webserver, as you might expect, you're going to be allowing port 80 and 443, if it's a box that isn't going to be exposed to the public Internet, you might just expose RDP ports and make it only available to a certain IP range for administrators in your environment. So you can control that and that's what this kind of acts like is the firewall policy. And you can have multiple Security Groups for a server and to some extent this creates a little virtual network if you will, so servers within a Security Group can communicate with each other. Important to understand some of the IP processing, so instances do get IP addresses automatically. They get a private IP assigned when you spin up a server, and they also get a public IP. And that public IP is associated with that server until it's stopped or terminated. So as soon as that instance is launched it gets a public and private IP address. The private IP address won't change while that machine is still running and alive. Its public IP can technically change in some instances. When you do stop a server the server itself, the instance is actually terminated. And when it's restarted they spin up another instance of it so you may get a new and different public and private IP address when you stop a server and then start it up again. EC2 is, there's some highly available components built in, obviously it's built to be redundant underneath the covers, but at the end of the day this is not a default highly available service. A machine if it tips over and you don't have a backup environment, you're out of luck. So need to architect for high availability when working with EC2 and we'll cover a lot of that in the remainder of this course.

Deploying EC2 servers

Deploying EC2 servers is pretty simple and straightforward, that's one reason I think people jump to this instead of waiting for weeks to get these internally. So deploying the server is as simple as going to the EC2 portal or using the SDK or using the API and choosing first your region and availability zone. So these are tied to an availability zone specifically, that's where you're deploying this to and obviously the overall region that contains it. So again if that availability zone goes down and you haven't replicated the information in servers elsewhere, you're application will be offline. You choose a server name, choose a key pair for the credential encryption, so how am I going to encrypt the credentials, you choose one of the key pairs associated with your account and then you use that key pair to get the password later to login. You select or create new Security Groups. So this will be where you define some of those firewall policies for what is the inbound traffic allowed and so forth. So that's where you set that up. Then you monitor the instance launch, this usually just takes a matter of minutes to get your server online and you can see it running, you can see some of the status changes within the console of what's happening as it's booting up. Then you can view some of the running server settings, IP address, a DNS name, the Security Groups, you can view all this information after the servers up and running. Then you can go ahead and connect to the instance if it's a Windows server, you'll probably use RDP, Linux Box, you'll SSH in and you can connect assuming you've allowed your IP address through that particular port via the Security Group. And then you can do things like attach EBS volumes if you need to or add software or install your applications and things like that, but once you have your server then it's up to you to treat it like any other server. And you simply deploy software as you see fit and get things running. Next up let's go ahead and deploy some EC2 servers, we're going to do one for our webserver, one for our queue processing server, and see what that looks like.

DEMO: Creating the IAM role

Okay let's build some servers. Now the first thing we're going to do is something I haven't talked about much yet, but it's one of the most critical steps for running this efficiently and securely in the environment. And that is specifically using identity and access management. So IAM is a relatively new service that manages access control to AWS resources. Kind of create some role based access. Why it's valuable here is that we can create roles that have access to other Amazon services. So that applications running in EC2 instances don't have to have credentials themselves imbedded in the app, the server starts under a role and so that means it runs under that role. Which means when it invokes services let's say in DynamoDB or SQS or any of these other things, it's running as that user and it's already authenticated. So instead of having to imbed credentials in config files to access these other services, you don't have to do that, much more secure, much more scalable. So instead of, as you've seen so far in our application, the web config file had to contain Amazon credentials to access key and the password I don't need to do that if I use IAM on the deployed environment. So let's first go to IAM. And what I want to do is create a new role, again not groups and users. I want to create a role and I want to add one that's going to have access to DynamoDB for the session state and I want to have one that has SQS so I can access the queue and pull and retrieve messages. I'm going to create a brand new role and let's give this a name let's call it WebServerRole. And I'll probably also use this for the queuing to be lazy but in reality you would want to set up as many unique roles so you don't over provision permissions to a particular server. And we'll click Continue. Now I have some choices here, you can already pick some out of the box ones and I want to first start with and EC2 role. So as you can see I want EC2 instances to call other AWS services on behalf of this role. And yes that's what I'm looking for. So I'm going to select that. Now I can choose a template. I have Read Only Access to a bunch of services, there's specific services I might reference here, instead I'm going to go ahead and create a custom policy. And type in my policy itself. I'll call that DynamoSqsPolicy and it takes the form of a JSON file. So what this does, and again you can build this up as well graphically I've already saved it off as a JSON file, but what you can see is that you pick actions. So what are you allowed to do? In this case we need, these are the permissions you have to have so that the session state provider works, have to be able to delete an item, describe the table, get an item, put an item, and update an item. And then this would point to the individual table that we've got in DynamoDB. And so this takes a form of a particular structure, it's looking at your account ID, the table name that you're allowing access to. And here is that session state table. Secondly I'm having another allow permission in this case deleting a message from a queue, deleting in batch, getting the queue URL, a few of these queuing operations and here's the name of our queue shapeorders. And I'm allowing access to that. So this policy is now saying that anybody who's running under this role and an EC2 instance, is going to have these permissions to those assets. Then we can go ahead and create the role. Now when we create an EC2 server we can actually say use this role and then it runs under that role and has those permissions. And now I no longer have to imbed any credentials in my application if I'm performing those activities. Next up let's jump into EC2 itself.

DEMO: Creating the servers

So here on the EC2 page we can see I don't have any running servers so far. And I'm going to go ahead and launch an instance. I'm going to launch an instance of my webserver and I get a choice of wizards that you want to walk through. I'm going to choose the classic wizard so I have some more fine grain control over what I'm setting up here. First thing I have a choice of is where's my AMI coming from. Is it a custom AMI I've built. Are they ones that Amazon recommends that are some of the most popular simple ones to start with. Community ones that others have created or market place where people have uploaded ones with package software built in. In our case we're just going to have a simple webserver so I'm going to start off with a base Windows Server 2012 image for our webserver. I'm going to have a single instance, this is just my webserver for now. And I'm going to choose a small one so I have a little more RAM than just the micro. And based on the region I'm in, again remember I'm in us-east, I get those availability zones. I could choose which one to run into if I chose to, I don't really have anything special or any I care about, so I'm just going to let it choose what works best. Then I have some Advanced Options, including if I want to turn on monitoring or very importantly for us which IAM role. So by picking this this server will now run as this role. I can turn on termination protection which make someone uncheck the box before they can actually terminate a server. So it makes it a little harder to do so. You can pick a shutdown behavior should it stop or actually terminate the server. It'll show you the storage device configuration where you can add volumes things like that. Add some metadata tags that you can access from within the server from outside. Picking a key pair, so this is how you secure in your credentials for accessing the server, this is how you can decrypt your credentials to login. And now we're choosing Security Groups. So I don't want any of the ones here so far, instead I want to create a brand new one. And so again this is a firewall policy remember, so what this should be is the WebRdpGroup and this will be for web or rdp access. And so I can create a brand new role here and I might say first of all for HTTP, I'm going to allow all traffic. So I have all sources allowed. Now for RDP I'm going to have obviously something more specific, I'm going to allow just my IP address to access this. So I'll plug in my IP address here, add that role, and so now this group is saying all traffic allowed over port 80 only my IP address is allowed to access this over 3389. So I like this group, WebRdpGroup, let's go with that. Continue. And I go ahead and launch the server. And so while that's running. I can see it going and it's going to go ahead and start and I'll take a few minutes. So while that's running I can go ahead and create a second instance of the EC2 Windows server, is that I want to use for the queuing server. So let's launch another instance. Choose the classic wizard. Once again I'll jump down and jump into a Windows 2012 Base Server. Once again 1 Instance, we'll keep this as a micro for now that's fine again you can mix and match things within your solution. I don't have a preference for availability zone. We'll Continue. Once again I do want this to run under the webserver role because this one also remember has SQS access. And so this will be the one that's pulling from the queue. Nothing fancy here for storage. No specific instance details I feel like setting. I'm going to use my existing key pair, but I want to create a different firewall policy now. Because I don't want to allow web traffic to my queue processing server. So instead I just want to come in and call this RdpGroup and I'm going to come in here and I'm going to choose RDP, again you can add any port numbers this just helps you out a little bit. And once again apply my IP address and add that role. So this server I can only RDP in, that's the only inbound traffic allowed. Click Continue. I can see my settings before I'm about to launch this thing. And I'm launching. And so now I've got both servers starting up, it does say running, but in reality it's running status checks and things like that so the servers not going to be available for a little while longer, but you can already see some of the settings come into play here. For an individual server, I've got a Private IP address already, Public DNS, so I've already got some values available. You can see the Launch Time, see that it is in the WebServerRole, what the Key Pair it is using, and so forth. So you can already see this one starting to run, running all the status checks and in a few moments it'll be available for RDP access. So what we've done so far then is create two servers. The next step I'm going to do is jump into the webserver and install IIS on it so I can actually host a web application.

DEMO: Connecting to servers

And again after a few minutes now our servers are mostly online I can confirm that's the case if I right-click I can now choose Get Windows Password and again you can see you can do some other things with this, but I can Get Windows Password. And if this pops up that means it's ready. So I have an encrypted password but I have to decrypt it. And this is where it's looking for my key pair. So I can choose my Key Pair file. Decrypt my password. And I know that right now this is my temporary or my actual password which will be for this server User: Administrator and here's this computer. Now I can also change that, but for now I can also do Connect. And what this does is download an RDP file that I can use to access this server. And what I want to do is I want to RDP into this server and install IIS so that I can run our web application. Here I'll plug in our decrypted password, you see this is connecting to AWS. And I'm logged into the box and I can use the administration interface to now install the IIS role on the server. Here I can add roles and features. I'm going to add a role to the single server. And specifically I want a webserver and you do have to make sure you choose all the different sub options. Specifically you do want to make sure you actually install ASP.NET, it's very easy to just walk through this wizard quickly and realize you didn't install ASP.NET and you end up with a nonworking webserver. Then making sure we're configuring the webserver properly with the roles we need. I'll just add a few additional settings like Windows Authentication just adding a few more pieces here. And simply finish the wizard. This is just setting up a basic IIS, whether I need all those features or not this at least gives me a nice running environment. Now with that installing by the time we deploy server IIS should be ready to go and that finishes this part of the demonstration of just getting the servers set up and in a few moments we'll actually deploy our app to these servers.

Accessing shared content

Before looking at how we deploy servers a quick detour into how do we access shared content. So how do I share data between servers at runtime? It's going to be a constant question you might have as you have larger and larger server farms, how are you sharing content? So you can use things like queues of course to message data between, that makes great sense for asynchronous durable messaging. You can use kind of a network file share pattern were you set up one EC2 instance with let's say EBS back storage and then expose a file share from that that the other servers can access for potentially shared content or files configurations. So you have one server kind of acting as a network share. You can use something like S3 and we've talked about that using S3 to store, let's say image files and other static resources used by a web app. And of course you can use persistent databases, like RDS or Dynamo if you're accessing master data or reference data. So it's important to think about how do I want to share data between servers so that these servers can remain stateless. So what are the different things I can do? And then you also want to be able to access common files during that provisioning boot up process. Again bootstrapping the server so that I may be pulling the latest application configuration file from a network file share or S3 or something like that. Again media files, I might want to access during boot up to pull into my virtual machine. Or even the latest application components, I want to grab the latest version of the web app that I've deployed as one place and whenever a server boots up I grab that and that's what I install on my server.

Deploying web applications

When we're deploying web applications we do have a few choices for how we might want to do that. Clearly we can deploy manually, right, I can access the shared location, I can put this into S3, I could even RDP into the server and just share my local drive, which I'll actually do in a few moments. And push the file up to the server, clearly that's not going to scale well, I'm not going to do that for every new server, that doesn't make sense, but as I'm creating my initial server maybe that would be a way I want to deploy. There's also automation frameworks, whether it's CloudFormation offered by Amazon for kind of creating templates and doing some automated deployments. Where you might pull files from a shared repository. You can use standard tools like Chef and Puppet to do automation and continuous deployment if you'd like to bake that in. You can use tools like the Elastic Beanstalk, which has multi-language support, so if I'm pushing .NET or Java or Node applications you can package into play applications often from within your own development environment. And we'll be touching on this in the follow up course to this one. Next up what we're going to do is actually deploy our web application. So we're going to take our web app, we're going to deploy that, we'll also deploy our queue app and then we're going to see by hitting this over a public Internet URL we're able to see our ShapeStore on the public Internet.

DEMO: Deploying the web application

So we're back here in Visual Studio and what we want to do is take our ShapeStore and our order processer and push those to the appropriate servers. So first thing I'll do is go ahead and publish a version of this website to the file system that has gets rid of all the CS files, kind of gives me a clean web app. So I'm going to click Publish. And I've got a PublishWeb profile that simply drops it to my temp directory in this folder. And I'll publish that. And that gives me the files now that I can go ahead and copy up to the server. So what I'm going to do is RDP into my server also notice for Local Resources I'm exposing my C drive, making it simple for me to send the file from C temp up to that server. So here we are in our server. I go ahead and open up the IIS console if I'd like to, because we installed IIS previously. And what I'm going to want to do is put the files in the IIS directory. Here under inetpub, wwwroot, and I want to go ahead and copy these first from my local machine. So as you can see I could access my local machine now via RDP. And I'm copying the ShapeStore application up here to my virtual machine, which I'll then deploy to IIS. So that application is now uploaded here, I'm going to go make a copy of it just so I keep it here as well. Make a copy under the wwwroot. And come into IIS and expose that as a web application. We'll convert that to an application, go ahead and Save it. Now before this runs we're going to have to do a couple of things. Right now this machine has no access to actually to RDS itself, so we created a policy for the role so it could access DynamoDB SQS, but I want to make sure that this server can access RDS, the relational database to get its data. So what I need to do is go to RDS. And what's really cool is that I can actually add EC2 Security Groups to the RDS Security Group. So here under Security Groups. You can see there's a default here, it's just my IP right now from managing it now. What I want to do is switch to EC2 Security Groups. So I can say the WebRdpGroup has access. And I also want to provide access to my just regular RdpGroup. This is the one that would be the queue server. And so what this is doing is saying any server in the Security Groups is now going to have access here. What this is really important means that instead of trying to do IP by IP which isn't going to work for an environment where servers are constantly being created and deleted. I can't try to manage an IP range or specific IP access to my database, instead anytime I simply deploy a new server to either of these Security Groups, they're going to have access to the server. That's a really, really powerful thing that I can do there. So that's authorized now. Now let's go back to the virtual machine. And the other thing I want to do here is I want to remove the credentials from the particular config file. Because I shouldn't need them anymore because I'm running under that role we created with IAM. So if open up the Web config and my text editor. What I want to do is remove all of the credentials again, not database ones because I'll need those, but what I can do is take the DynamoDB session provider for example and I no longer need these credentials. Because I don't want these imbedded in my config file so in the real world I wouldn't even upload this even to the server itself. I would simply clean this out before I sent it up here. So these are the keys being used by both the session state provider and the queue piece where I published to the queue on a new order. I'm removing these and the SDK is smart enough to go ahead and get these values from the server itself as part of the role. What that means is when I start up a server under a specific role I can actually access metadata through a URL that pulls in some temporary credentials. Let's go ahead and show you what those are. And so if plug in a very special URL that every server has enabled, so it's a local IP address, it's just pulling some data. You hit latest/meta-data/iam/security-credentials and whatever name of your role is. It pulls back some temporary credentials. And so the SDK is smart enough to call this when it doesn't see credentials. Gets those credentials that are associated with that role and that's what it sends to the service. So we've gone ahead and removed the credentials from here. And because I always forget this make sure when you do add the roles that you've ASP.NET from a development perspective. And make sure it adds the right extensions as in mime extensions and I forget this every single time. So this time we'll go ahead and make sure we install that before we run the application because otherwise it'll fail and my demo looks terrible. So now we'll go into the website and try to start it up. Hey and sure enough it went ahead and started the server up. So you can see loading everything up, all my images, this is all running in my Virtual Machine. So obviously it's able to access the database through RDS, can store session, I've got everything available here in the cart if I need to. Proving that the roles work. So if I add something to the cart and I see the cart again. This is in session. So that means this is talking correctly to Dynamo with no problem. Now let's hit this over the public Internet connection. So here in EC2 we can look at our small instance and we want to grab the DNS address. And we should be able to access our web app because we've opened port 80 to the public Internet. So by just hitting the raw address we get IIS8, that's a good thing. And by hitting the ShapeStore sure enough this is a public Internet address and it's pulling up the appropriate IP address. It's showing me the IP of the server and here it is. So I'm running the app over the public Internet deployed to a cloud server.

DEMO: Deploying the queue application

Now let's deploy our order processing application. I went ahead and did a release build of this application. I'm going to copy it to my temp directory. Also make sure you have all the references, so make sure you're AWSSDK is copied local and everything. So you've got everything you need. I went ahead and pushed that to the temp directory and logged into my virtual machine. Within my VM I went ahead and copied the temporary file to my desktop. And with that on my desktop what I want to do is create a shortcut to the application, which I'll put in the Startup folder. So in real life this would ideally be a Windows service that runs, but in this case we can survive with just having a little app that runs by itself when you start up the server, but ideally this would be a Windows service that would just run constantly. So I'm going to cut that. And I'm going to want to go put that in a specific folder that runs at Start up. And you're going to find that at C ProgramData Microsoft Windows start menu and Startup. And we'll paste that in there and that would run when the machine starts up. So let's go ahead and see if we can start this right now. Sure enough it's pulling the queue. So if it finds anything in the queue it should be able to talk to the database. So every 10 seconds it's going to pull the queue, let's go and head back to our browser and add an order and see if we can see this show up. Now let's first go look at MySQL and confirm there should only be one order in our database from our testing before. I'm here in MySQL, here's our orders table, this is the one looking at RDS, this is one logged into the cloud. Only a single order in here right now. Here we're at our Shape Store hitting our public Internet URL. Nothing in our cart at the moment. Our sessions timed out. So let's go ahead and Login. We're logged in we can see the Logout button, let's go ahead and add some things to our cart. And then we'll check out. And so when I click Buy now it should drop it to the queue. And when I switch back to my machine here it should uncover it during one of its pulling intervals. Sure enough we just saw a count of 1. Hopefully it's going to be able to write to the database, delete the message, and continue pulling. If I come over here to MySQL and refresh my database, sure enough I've got a new order. So it successfully, my cloud system is now successfully all linked together. I'm able to push things through the queue, read things from the queue, update my RDS database, now everything is running in the cloud. And there are no credentials, minus my database credentials, but there's no credentials now imbedded in my application. So I don't have to have credentials for RDS, or I'm sorry for SQS or for DynamoDB, that's simply using the instance level roles we've provided. So this is really kind of a cool scalable secure application we've deployed so far.

Creating and using AMIs

I've talked about AMIs briefly so far, but let's dig into these a little more. These are the Amazon machine images and these are going to come into play much more, especially now that we have working servers. So these are a critical asset if you're working in the cloud. They act as a unit of scale for an application. And in essence, when you want to scale an application ideally you're not scaling up as much as you're scaling out. Which makes AMIs the unit of scale because I'm constantly adding more servers. So a very, very important piece. I can create these in Administration Console, via the API, via SDKs, so very simple to set up. Prepare the server with all your software and settings first. So add any necessary software, in our case we've got our web app running, we've got our queue server app running, everything's ready to go. It's technically a finished server that I can take a snapshot of. Then in the console if I'm using that, I can simply click a "create image" button. Give the AMI a name, because it's going to persist that template so I can access it over and over again. Provide some sort of description, so it's easy for me to understand what this is later. Then the instance will be shut down, a snapshot is taken of the image, and the AMI gets registered. And then it's accessible anytime I want to create a new virtual machine I can pick from one of my existing AMIs. Next up we're going to do a demo of this, we're actually going to create AMI images of both of our server types, save and persist those so that we can use them later.

DEMO: Creating and using AMIs

I'm back here in the AWS console and what we want to do is create AMIs from our running servers. So I can jump to my running instances. I've got my two running servers. My small one being my webserver, my micro one being my queue server. And even added tags just to make life even simpler for myself, so server type web, server type queue, you can do the same thing if you'd like. So for my web one I can right-click and I can do Create Image (EBS AMI). So I'll do that and we'll give this instance a name. And this first one might be ShapeStoreWebServer, I can technically choose not to reboot the instance while it takes a snapshot to create the image, but again you get a warning letting you know the file integrity might not be in place. Because things could be written and so forth and you might miss those on the next server. So we're going to go ahead and allow it to reboot, that's fine. Here you can also choose root volume, what type of volume you want it to be, provisioned IOP, and so forth. You can specify a lot of the settings you'd like to do. And then simply click Create. And it's going to go ahead and start creating the work to create the AMI. And we can do the same thing for the micro one, we can right-click this one, Create an Image (EBS AMI), and again this one's going to be a ShapeStoreQueueServer, because it's the one that processes data from the queue. And once again I could add more storage, I could do all sorts of things when I actually create this image, or go ahead and just click Create. And I can monitor the progress. Under AMIs you can see it's starting to work. So things are happening, it's creating a pair of AMIs that I'm going to be able to use momentarily to create new servers. After a few minutes we've got our AMIs up and running, we can see that they're no longer in a processing state. I can see some details about these. And then from right here I can go to instances and launch instance, but right from here from my AMI list, and you'll see these are filtered owned by me all images. From here I can go ahead and do launch. And I walk through the same wizard process, I can decide to do a Micro instance in this case or a Small instance. How many do I want, same question about availability zone, this could be where I start to distribute my servers around different availability zones. Once again, very, very important I want to pick my webserver role here to make that I'm still going to run under this, that's the only thing that if I don't set this this templates not going to work correctly. Pick my key pair. Walk through my process. Once again I want to pick the right Security Groups, I've got the right firewall access. And then launch. And once this is done launching the only thing I should have to do now is hit the URL. It's got all the software it needs, it's got the webserver up and running. I should be able to hit the URL, get a different IP address, and see that I now have two servers running the same software in the cloud. Finally our AMI is completed and we have a brand new server. We see now we have two of these small ones. I can use this particular address to now hit this different one in the browser. And sure enough it resolves it. So this is a different address. We've got different websites here. You see different IP addresses. Both of these running in the application. So I've got two webservers both running, both accessing the same shared session server, and all of that. So a really neat way to quickly spin up servers that can now provide additional capability as I scale horizontally.

EC2 best practices

EC2 is a great service for dev test, production apps, but you do want to make sure you keep some best practices in mind. First and foremost, designing for failure of cloud infrastructure expecting things to fail. Building servers and applications that expect the stateless environment. So in this case understanding again, it's not inherently highly available under the covers. So you do want to make sure that you've built an app that can scale across horizontally multiple machines. Don't rely on instance storage, again it's great for scratch data, it's great for temporary processing, definitely take advantage of it, but don't rely on it for anything that might need to be persisted somewhere else and be able to survive a machine failure. Go ahead and choose the right instance size for your workload. You might want to make sure that you've got webservers that have more compute resources. Cache servers that need a lot of RAM. Database servers that have a lot of RAM and CPU. So choosing the right instance size, not just falling to a default and not over provisioning. You can size these things up and down later, so no reason to behave like traditional IT where you provision for the absolute maximum capacity and then probably run idle a lot of the time. Instead provision what's a reasonable starting point and understand that you can scale up and down or scale out. Make sure you're looking to see if your app is CPU or RAM bound or storage constrained where you want to make sure you provision a lot up front. Understand what sort of app you have and provision accordingly. Create AMIs for specific workloads, don't try to create one AMI that serves every purpose and can be a database server or a webserver or this or that. Really you want to make sure you're creating specific AMIs and in some cases some companies even create AMIs for every build. And make sure they've got that. So think about how you want to do that. Use IAM for role based access, can't stress that enough. It's a really cool functionality and if you're using SDKs, the SDKs know how to work with IAM to pull those temporary credentials to access the service. So really in almost every case you shouldn't be imbedding any AWS credentials in your app deployed to EC2. Instead use these IAM roles to make sure that your accessing these based on permissions you've set up in centralized policies versus having to imbed credentials on the servers. Figure out what you're best AMI deployment strategy is. Do you want to do the software-packed AMI, where everything is on the AMI, it's finished complete, it's got all the software there already, that's going to be very fast to deploy because everything you need is there. Or you want to bootstrap a basic AMI that's got the basics of your web app, but then after it runs maybe you want to go grab configuration files from S3. Maybe you want to grab the latest build from S3 and deploy it. So you have to think about what's going to be your best technique. Bootstrapping might be the most flexible because I can mix and match components on the fly, change my scripts, I don't have to constantly change the AMI itself when there's a new software build. Really just depends what's going to be the best fit for you, what's the best process you have already for continuous deployment, do you have that process in place, or do you want to be able to mix and match and do a bootstrap process.

Using Elastic IPs

Something that's complimentary to EC2 and one to focus on in this module specifically, is the idea of an Elastic IPs. So we talked about earlier when you provision a new server you do get the public IP, you get a private IP, if I do stop the server and restart them I'm going to get different values there. So, Elastic IPs give me something a little more flexible. It gives me a static IP address. That's a pretty powerful thing. So I've got a static IP address and I can rapidly remap that to other instances. So this is where it's pretty cool that I can have a server running, tied to an Elastic IP, maybe I'm going to run some maintenance on it. Maybe it actually fails, I can swap that Elastic IP over to another server and all of a sudden more or less stay up without any interruption except the very short period of where that remapping occurs. The problem this solves is as doing a dynamic DNS update obviously takes a while, this has to cascade through DNS providers. If I say, you know what here's the DNS address for this server, opps here's another server that now replaced that one, it's impractical to assume that that switch is going to happen quickly. So Elastic IP gives me a fixed IP address DNS entry that then I can point to various servers underneath. So this replaces the instances public IP address, as soon as you attach this it releases the other one and only uses this Elastic IP as its address. When I do disassociate from an instance, so if I remove the Elastic IP it does get re-IPed with a new public IP. Interestingly enough you're not actually charged when they're in use, you're actually charged when it's not in use. So when you're not using one but you've reserved one, that's when Amazon charges you, mainly because IP addresses are somewhat scarce. I think it's the IPV for ones, and so you're being encouraged to use the ones that you've provisioned. You are limited to five Elastic IPs per region, so you do want to make sure you're responsible with these and only using the ones you need. You can create and assign these in the Admin Console. It's very simple and straightforward to do that from the Admin Console and set these up and do that from there, we'll be doing that in a demonstration in just a moment. Now it is important to understand this is not load balancing, so don't confuse the two. This would be in a case where maybe I don't need the load balancer, but I do want to have a fixed IP for a simple webserver I've got running in the cloud. Or I've got other services where I want to attach this IP address to, just don't replace this mentally with the idea of a load balancer, this simply gives me a fixed IP address that I can attach to a server.

How is failure handled?

So if we think about how failure is handled in an Elastic IP world. If I have people coming in through the public Internet using an Elastic IP to attach to a webserver. If something goes wrong or let's say I use an AMI to build a new webserver and I want to perform some maintenance on that first one, I can take that offline, repoint the Elastic IP, and then all of the sudden everything's going to keep working. You know in some case if I'm doing manual maintenance and I take these servers offline on purpose, there might be zero interruption for this.

DEMO: Using Elastic IPs

Alright so let's go ahead and add an Elastic IP. When you're in the EC2 dashboard you'll see Elastic IP is down here on the left hand side under NETWORK & SECURITY. At the moment I don't have any allocated. So let's choose Allocate New Address. I'm not using the virtual private cloud, this is in the public cloud so I'm choosing Yes let's allocate. And sure enough it gives me an address. Now I'm going to choose to let this age for too long, because we're just doing a demonstration here. And so once I have this IP address I can right-click and choose Associate. And now I can choose one of my EC2 instances I want to tie to this. Let's go ahead and make sure I know which one I'm dealing with here. And we'll choose this small one, instance a44 starting within the prefix there. We'll Associate with a44 and choose Yes. Once that's done you'll see I do get a public DNS, now again as I swap this between machines this is going to stay fixed. And that server also is going to get its own new public IP address tied to this. So now how this should work is I should be able to take this address. And plug this in and get the Shape Store which I did. Now again this is pulling the internal IP of the server, but you can see I'm still pulling that using an Elastic IP address. So I'm getting that with a fixed IP address. And seeing that come through. Now what you should see is that this one's pointing to this IP under the covers. If I disassociate it from this instance and tie it to the other instance, which let's look at its prefix, the other small server is 71ee. And I can go ahead and Disassociate or actually even simpler I'll go ahead and just right-click and Associate with a different server. And I'll click Yes. Swap that, again my IP address shouldn't change. And when I refresh technically this would be a different IP under the covers, but it actually did get assigned to actually a new private IP. So that's actually why yours did remain the same, is that value shouldn't change. Because that private IP is again associated with this particular server. So that's great, so I was able to swap my servers completely under the covers. By swapping an Elastic IP address and meanwhile everything just kept running. So there's no interruption if I'm providing support or maintenance, none of that matters. Likewise by accessing through the public DNS. SO this is a great way again to give me that fixed value, I'm able to quickly swap this between servers with zero downtime and have a nice way to allocate addresses Associate and Disassociate all through the control portal here.

Elastic IP best practices

You've Elastic IP in practice now, let's talk about some of the best practices. So first again recalling this is not for load balancing, it's more about high availability. It's making sure that you can keep servers online, you could make sure you're swapping things behind the scenes, but you're providing a fixed interface outbound. So that people don't have to change their connections or worry about that. Great for rolling updates, great for small environments, again you can even use these for different sorts of servers besides webservers. So it is great to know you could have a fixed public IP. Cloud management provider, RightScale pointed out a great best practice here which is aging in Elastic IP. Because in essence an Elastic IP could have been used by some else who then released it. So if you immediately map to it you may end up with weird traffic coming from somewhere else because DNS hasn't flushed yet, if you've mapped it to your own public URL. So you might want to request your Elastic IP, let it sit for a little bit, 24/48 hours, whatever, and then simply use it after that. You're going to want to choose the right instance type to use with this. It's going to be a best fit for things like public URLs, but again you can use it for other things. If you're hosting your own database, you may decide to use this so you don't get a new public IP address on every stop and restart. So Oracle even recommends using Elastic IP to get a static IP for Oracle DBs in AWS. So not necessarily just for webservers, but think about what's the best fit. And again use these with constantly changing environments especially if you're just working through things, taking servers on and off, but wanted to have that fixed public interface, it's a great fit for that. Be careful though not to steal Elastic IPs, when I'm starting a new server I can go ahead and assign an Elastic IP even if it's already in use. And therefore steal it from the server that was already using it. Now you might do that on purpose, but just be careful you're not told not to do that. So you can technically take an in use Elastic IP when you startup a server. Make sure you understand the public versus private, so when the EC2 instance, if I have an instance that queries that DNS name, that's tied to an Elastic IP. The DNS server returns the internal IP address of that instance, not the public one. So that's pretty good technically because I don't pay for public bandwidth if I start routing traffic to a server that has an Elastic IP attached, but you do want to make sure that when you're doing Security Groups or others that the internal servers when they hit that DNS name, they're going to get the private IP not the public IP. So if you accidently allow access to the public IP and wonder why things don't work, it won't work because internally it's all looking at the private one. So just understand the nuances between public and private IPs and Elastic IP.

Summary

In summary we covered a lot of stuff in here. In the reference architecture we finally started to pull all these pieces together where we've been able to deploy virtual servers that have their role of running our web application, running our batch processing, and connecting to all these other complimentary services. Like database and storage and so forth. EC2, very powerful, highly available, virtual machine providing service, again an individual virtual machine not's going to be super highly available. You've got an architect around that, but the service itself available globally, available to easily deploy lots of servers quickly, you can build very highly available solutions with EC2. We saw how you can deploy them, picking instance sizes, picking where you want to stick it, sticking which identity and access management role you want to attach. We looked at deploying web apps, you may do this programmatically, you may do this manually depending on what's the best fit for the tools you have available. AMIs are the unit of scale, horizontally. So creating and using them we saw how simple it was to create an AMI, spin up a server from one, and then instantly be ready to go. A lot of best practices for EC2, about how you use and provision servers for the right level of capacity, securing things well with Security Groups. Elastic IPs are a really powerful way to get fixed IP addresses in a relatively transient environment, where IPs are getting added and dropped constantly under the covers. If you need a fixed IP great to use Elastic IP. And it helps with failure or even rolling up dates and things like that. Then finally we looked at some best practice for Elastic IP including aging your IP to make sure you don't end up with weird traffic coming from an old consumer of the IP address. Hopefully you enjoyed this EC2 module, next we're going to jump into some additional things around load balancer for how you really create a highly available EC2 environment.