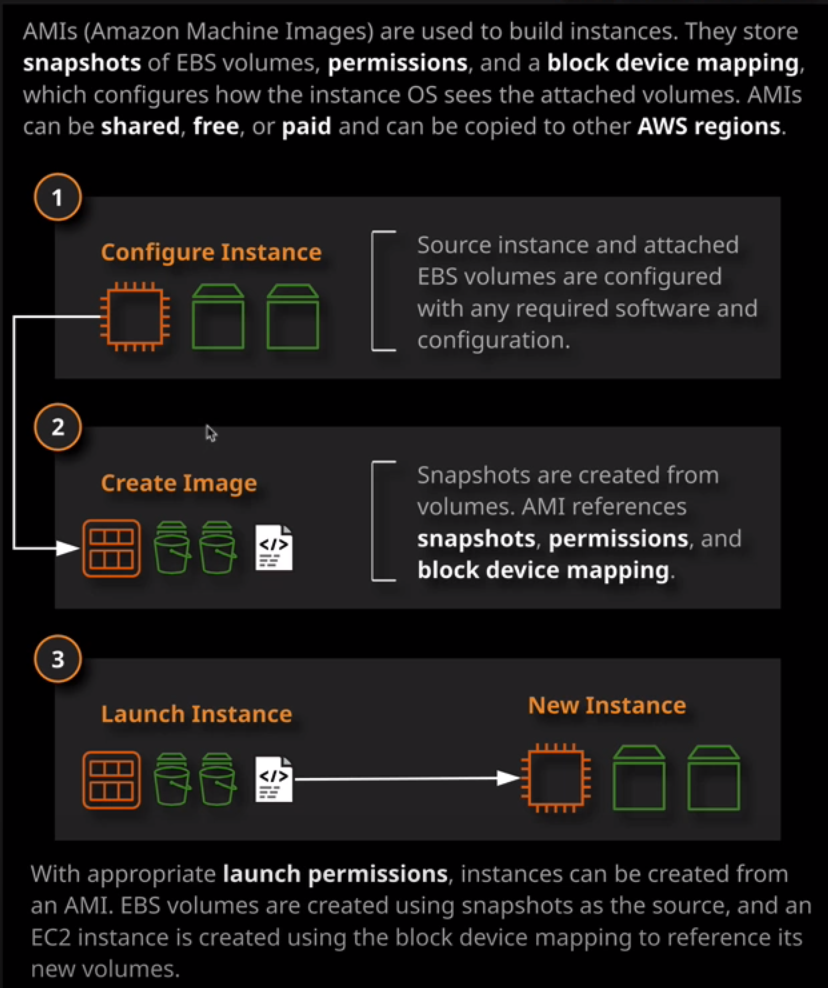
**AMIs known as Amazon machine images**  
  
An AMI is an object or container, and it allows you to launch EC2 instances.   
Two different types of AMIs. The first type is **instance store backed AMIs and these are used when the root volume of the instance doesn't use EBS.** Now they're pretty rare these days, and you tend not to experience them in production real world usage. The second type of AMI are **EBS backed AMIs** and these are used when the root volume of the source instance does use EBS.



EBS backed AMIs

Now when you're creating an AMI you're creating it generally from an existing EC2 instance.   
The AMI contains a number of things that are required to launch a new instance and these are based off the fact that you making the AMI from an existing instance. So because your source instance has EBS volumes, your AMI will need volume snapshots, and so an **AMI create snapshots and references those within its configuration**. In addition to that, it contains what's known as a **block device mapping**.   
Now every operating system, whether it uses directly attached or network attached storage, uses what are known as device IDs. So these **device IDs are the unique identifier for the device**. It's how your Windows or Linux operating system, knows to mount a particular drive or a particular volume. So **because this AMI has snapshots of these original EBS volumes, the block device mapping essentially stores a link between the snapshots and how they're mapped to the new instance.** So when you launch that instance, it'll go ahead and create new volumes, and it will know how to map them to the instance so they're accessible to the guest operating system.

Additionally, an AMI contains **permission**. So who can use that AMI, who can use it to launch an EC2 instance. **The best practice to create a good AMI is that you shut down the instance**. I'm going to go to instance state and then stop an AMI create snapshots and included in those snapshots is a snapshot of the root or system volume of the instance, and it's always best practice for consistency reasons to do that while the instance is in a stop state. **So when you're creating an AMI you need to make sure the instance is in a stop state to ensure that consistency of the root volume.**

**Block device mappings**. It's a list of all of the volumes that are attached to this source instance, together with their device ID. This device ID is what's presented to the guest operating system so, Linux, in this case, it's how Linux knows to map the individual volumes. When I'm launching a new instance from this AMI this block device mapping is how EC2 knows how to structure the resulted instance. So how do you know which volumes to attach and how to attach them so which device IDs to use. This should generally look exactly the same as the source instance. You are able to add additional volumes to this AMI but generally you're creating it in AMI to be in exactly the same state as the source instance. So in this case, I only have the single attached volume, the system volumes. So I'll go ahead and create the image. Now, at this point, what he see two is going to do is go ahead and create snapshots of all of the volumes that are attached to the EC2 instance that I just configured in that block device mapping. So if I go to snapshots, we'll see that we've got a pending snapshot for this 8 GB system volume that's attached to this Linux instance. So the first step in the process is to create the snapshots so that's these snapshots that are referenced in the AMI. Once the snapshots are created, the AMI will be created, and then it'll be good to use.

**The AMIs are essentially a container which references the snapshots that the original source instance had as well as this block device mapping.** Now, in addition, it has **permissions**. These permissions are launch permissions an AMI could be in one of three states. **It defaults to private, which means only the owning account.** So my account can access and use this AMI. I can change it to **public**, which means that every AWS account will be able to freely search for this AMI and launch instances from it or I can use a middle ground which is set to **private but with individual AWS accounts white listed** so that they can use this AMI to launch instances.

What I've done is have preconfigured a source instance from that source instance, I have created an AMI, including snapshots, permissions, and the block device mapping, and I've used this AMI to launch an additional instance. This example just creates a simple website with some cat pictures. It uses the user data to build it, pulling these images down from a GitHub repository. It takes seconds but imagine if the process that we used to build our website to install our application, to copy in our data, and to perform additional configuration took 30 minutes. **By creating an AMI with all that worked on upfront, it saves us that 30 minutes every time we deploy it and architecturally, that's known as AMI baking or AMI prebaking. You essentially do the work in advance. You do all of the extensive installation and configuration and then you prebake that into an AMI. It can save you significant time if you needed to deploy hundreds or thousands of the same application.**

**Another advantage using AMIs is an architecture that's called immutable architecture. Immutable architecture is a technique where servers, in this case EC2 instances, are never modified after they're created.** Imagine that we have a corruption in this Cat Hall of Fame website rather than connecting to it, performing diagnostics, fixing it, and hopefully getting it back into a working state. We could just terminate the instance and provision a brand new one from this known working AMI**. If something breaks, you delete it and create an identical version of that thing. Immutable architecture generally means that once provisioned, things don't change. If you want to make a change, you make a different AMI a version two and you deploy that new version of that IAM. IAMs are generally used either to perform base installations or when you're using an immutable architecture.** Generally, organizations have processes that every time they release a new version of their application, they create a brand new AMI with that new application version prebaked into it and it just means that whenever they need to deploy that application, it's in a known working state. That's immutability. You don't change it afterwards, you don't adjust things. You create a new version. You test that new version and then you use AMI to deploy across your entire infrastructure. **Immutability is a great way of ensuring a perfect clean infrastructure, which is free of bugs.**

Now AMIs can also be used for **scaling or high availability**. You have a feature of EC2 known as auto scaling groups and these could be used to deploy and scale EC2 instances based on load, and you can utilize AMIs to deploy new instances using a prebuilt application that stored inside the AMI. So it saves time when you're provisioning instances. Imagine if you've got an application that's running its business critical. One of the EC2 instances fails, and you need to replace it. If building a new server took five or 10 or 20 or even 45 minutes to build using your normal application process, then you wouldn't be able to recover for at least that time period. **If you prebake that into an AMI then you could launch a new instance from the AMI within minutes.**  **AMIs can be used where you've got complex applications installations or when you're using an immutable architecture and you want to deploy rapidly but AMIs do have their downsides. With AMIs, the config is built in. It's not something that you can really customize based on things like instance size or availability zone or subnet or IP address.** That's where bootstrapping comes in handy. Bootstrapping is when we use the user data to build an EC2 instance.