

AWS Intensive

WELCOME!



John Kidd



Develop Intelligence

A close-up, artistic photograph of a glowing incandescent lightbulb. The bulb is illuminated from within, casting a warm, golden-yellow light. The filament is visible as a bright, wavy line. The background is dark, making the lightbulb stand out. The text 'VPCs' is superimposed in white, bold, sans-serif font over the center of the bulb.

VPCs



Infrastructure philosophy

- The idea behind creating resources in AWS is that most of your infrastructure is **ephemeral**. You should be able to create it and rip it down at will.
- Things like databases (MySQL, Postgres, MSSQL, etc), servers (EC2 instances), and services (Kinesis streams, SQS, etc) should be, by nature, impermanent.
- BUT- there **are** permanent things that need to be set up....specifically two:
 - Your network (the VPC)
 - Your data (the S3 bucket)



Ephemeral vs permanent infrastructure

- So there are two ways to think about our infrastructure- the **permanent** infrastructure- made up of our data storage and network, and the **ephemeral** infrastructure- made up of our resources and microservices.
- In this upcoming section we're going to set up our **permanent** or **foundational** infrastructure first— which is a good thing to do when you are creating a stack: where does your data go and where does your code live?



Virtual Private Clouds

- So yesterday we went over how to create unique resources using terraform files. We went over a very **basic** terraform setup for a single resource deployment.
- Obviously for large scale deployments of complex applications we'll have to have a lot more than a single ec2 instance running (though it will work fine for a simple wordpress website)



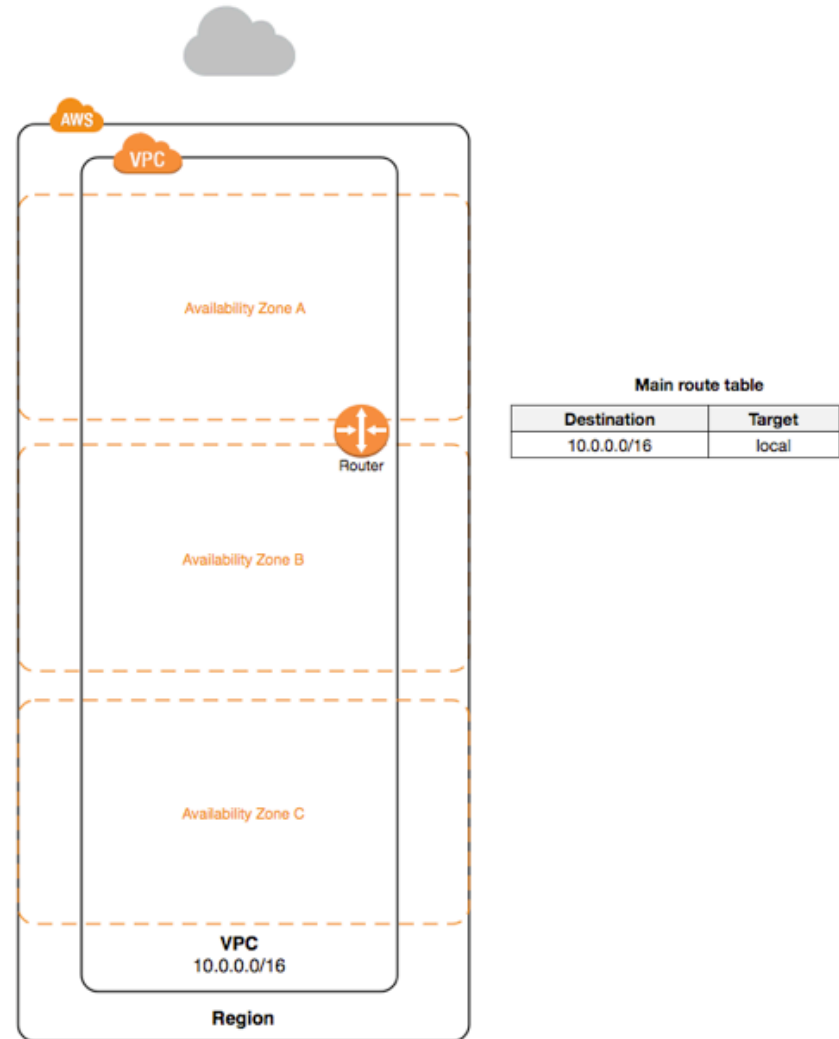
VPC access

- So foundational to any stack is the network that it lives in. **Virtual Private Clouds** in AWS is how we create these networks.
- VPCs isolate resources from each other whilst allowing for communication between resources within the same VPC.
- You can also whitelist/blacklist IPs, create unique security groups, and mount assets within the VPC to improve performance.

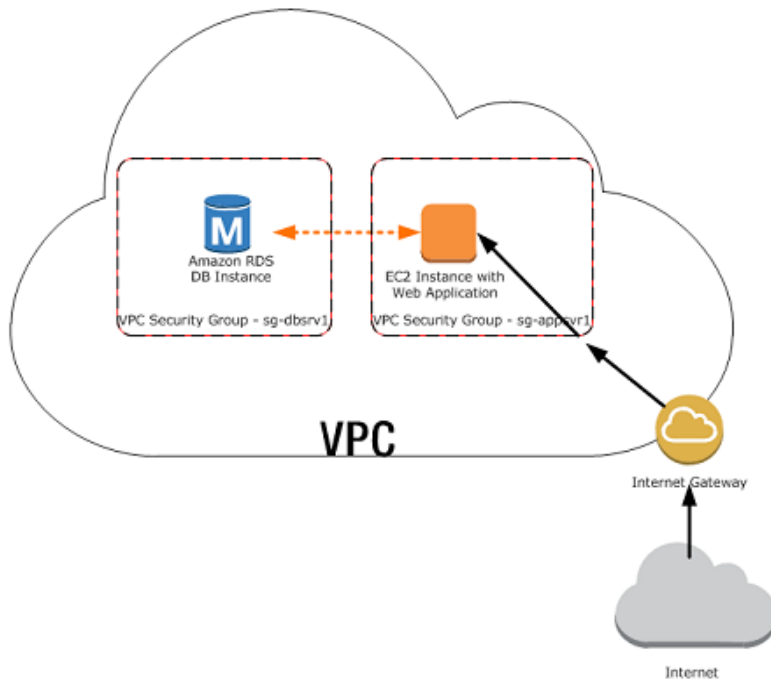


VPC Availability zones

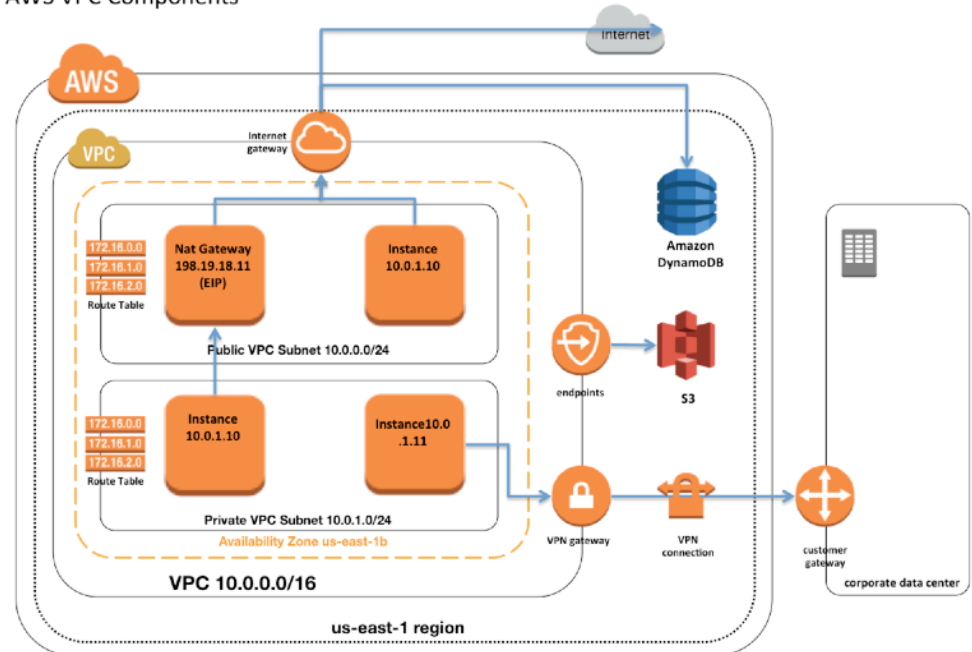
- A single VPC must have the ability to stretch across multiple availability zones in order to meet the requirements of a **Well Architected Framework**
- So a single VPC acts as your network and then within it we have multiple IPv4 addresses that make up the CIDR block



VPC

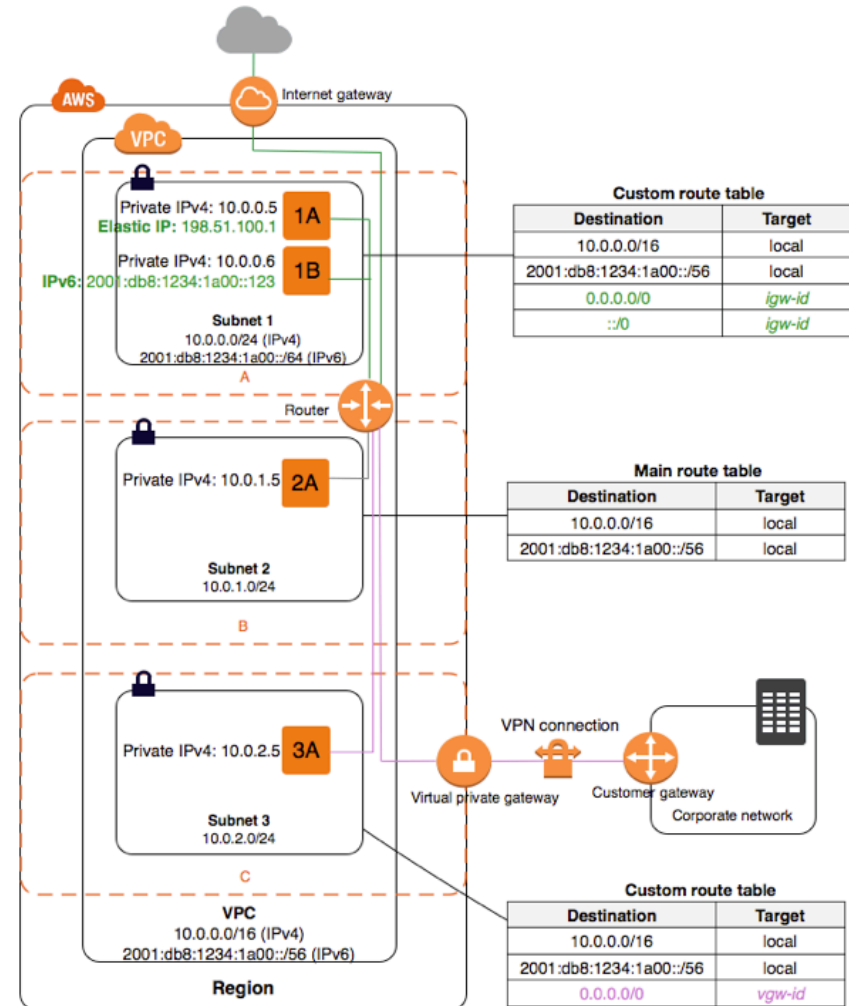


AWS VPC Components



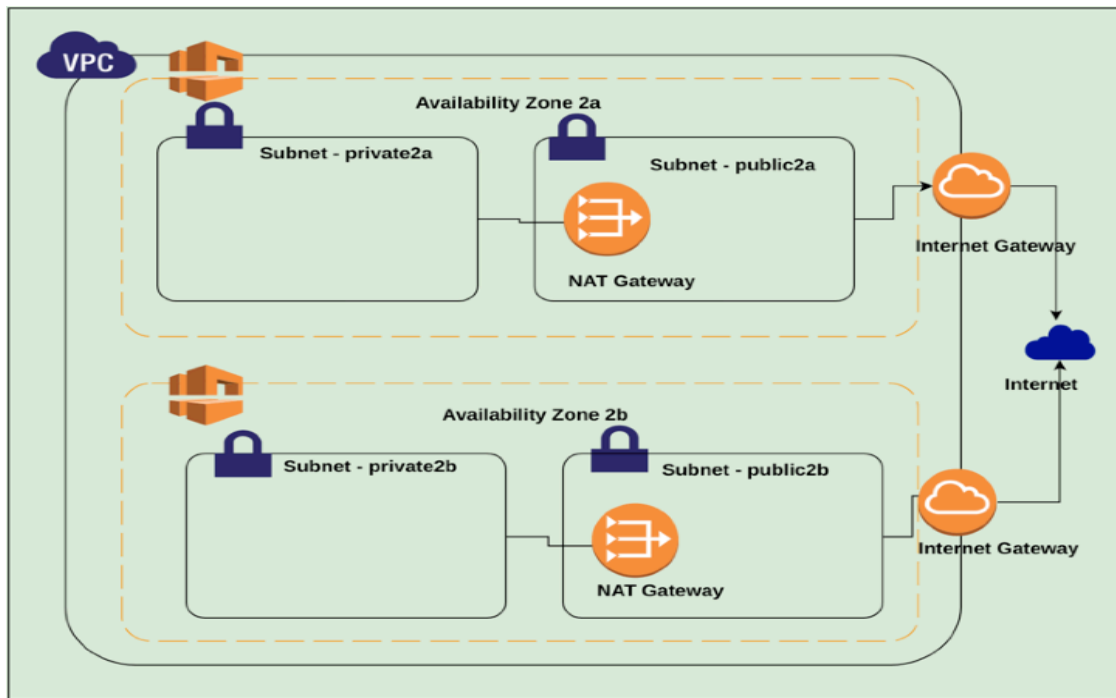
VPC (continued)

- In the example on the right we have 3 subnets in a single VPC:
 - Subnet 1 is public (accessible to the internet)
 - Subnet 2 is private (accessible only from within the VPC)
 - Subnet 3 is within a separate site VPN connection (Site-to-site in AWS terminology)



NATS in VPCs

- We **can** route data from a private subnet to the internet via a Network Translation Address (NAT) gateway that sends data through a public IP



VPC Subnets (continued)

- A VPC is denoted by a **subnet mask**. For example, when one says VPC-x is 10.123.0.0/16 , that means any instances inside this VPC will have an ip 10.123.X.Y where X and Y can be anything between 2 to 254. A VPC can have following global components:
 - A DHCP option set (server that assigns dynamic ips)
 - Internet gateway (will come to this shortly)
 - One or more subnets
 - One or more routing tables
 - One or more network ACLs



Creating a VPC and subnets

- When you create a VPC, you must specify an IPv4 CIDR block for the VPC
- The allowed block size is between a /16 netmask (65,536 IP addresses) and /28 netmask (16 IP addresses)
- For the purposes of this class we will be going with a 10/8 prefix so:
 - 10.0.0.0 - 10.255.255.255 (10/8 prefix) will be our available ip addresses in all of the exercises going forward.



Routing Tables

- **Routing tables** contain a set of rules, called *routes*, that are used to determine where network traffic is directed.
- So in the below example:
 - our 10/16 IP addresses go local
 - our 172 IP addresses to a peer to peer
 - and all others to an internet gateway (igw)

Destination	Target
10.0.0.0/16	Local
172.31.0.0/16	pcx-1a2b3c4d
0.0.0.0/0	igw-11aa22bb



VPC and routing tables

- VPCs come with implicit routers and come with a main route table that you **can modify**:
 - You can add custom routes
 - You can replace the table with a custom table (though you cannot delete it and not replace it)
 - If you have multiple CIDR blocks you can customize the routes as to where they go
 - You can add multiple route tables
- We will not be doing any of this (nor should you have to generally unless you get to a very complex architecting system)



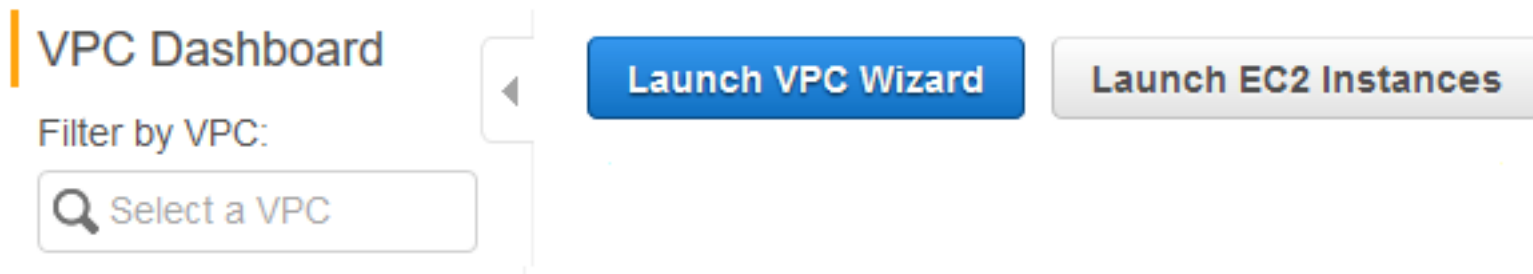
VPCs and Security Groups

- **Security Groups** within your VPCs act as a virtual firewall to any of your deployed assets within a vpc
 - You can have up to five security groups PER instance within a VPC
 - Security groups work at the **instance** level and **not** at the subnet level (that is controlled via the route tables)
- Security groups are pretty straight forward and from personal experience: 9/10 times when a developer cannot reach a resource: it's the security group.



Creating a basic VPC in the console

- So we're going to create a very basic VPC here with the console in order to demonstrate how it's done (in the lab we will be doing this in terraform but it's good to see how we accomplish this).
- Navigate in your console to the vpc area (just do a search in the services box) then launch vpc wizard.



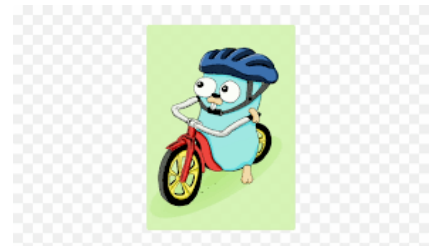
Creating a VPC in the console

- Choose a **single public subnet** (again- depending on your needs this might or might not be the case but most of the time you will want some sort of public access)
- The rest of the options are all customizable but **generally** and *with very few exceptions* you should let AWS choose them. That being said- it's important to understand them...



VPC from console

- **IPv4 CIDR Block:** You can customize this if you really want to look at bit ranges outside of the 10/8 prefix (most of the time just stick with this)
- **Availability Zones:** Again- customizable but not usually necessary unless you are really particular about where your VPC subnets live. AWS will choose by default
- **Service Endpoints:** This one is interesting as it can replace NAT gateways and allow subnets to communicate with various services



VPC Deployment

- **Enable DNS Hostnames:** Basically means that every ec2 instance deployed within the VPC will receive a DNS hostname.
- **Hardware Tenancy:** This will allow you to decide if you want to have a dedicated piece of hardware running your VPC (and resources). No idea why you would want to do this but the option is there.



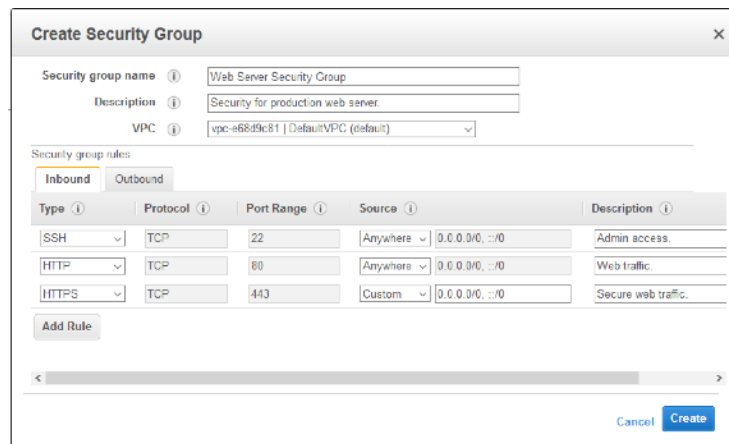
Launching your VPC

- Go ahead and launch your VPC now. It will take a minute to complete. Once done take a look at the **Route table** in the **Description** tag for your VPC
- Notice in the route table that you have two routes established- one reserved by AWS and a second one going to IGW or "internet gateway"



Security Groups

- The next thing you'll want in your VPC is to control access to anything you deploy in there (ec2 instances, docker containers, lambda functions)
- You do this via **security groups** which give you a great way to control access to ports on various resources.
- So on the left hand side of your screen click on **security groups**



The screenshot shows the 'Create Security Group' dialog in the AWS Management Console. The 'Security group name' is 'Web Server Security Group', the 'Description' is 'Security for production web server.', and the 'VPC' is 'vpc-e68d9c81 | DefaultVPC (default)'. The 'Inbound' tab is selected, showing a table of security group rules.

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Anywhere 0.0.0.0/0, ::/0	Admin access.
HTTP	TCP	80	Anywhere 0.0.0.0/0, ::/0	Web traffic.
HTTPS	TCP	443	Custom 0.0.0.0/0, ::/0	Secure web traffic.

Buttons at the bottom include 'Add Rule', 'Cancel', and 'Create'.



Security Groups (continued)

- So obviously the most secure way to run your stack is to only open up ports used by whatever application you're running....SO some examples:
 - If you are running a MYSQL instance- you want to open up port 3306 on your security group
 - For Postgres: 5432 (or 5439 for redshift)
 - For Node: 3000
- And obviously for web apps: 80 and for ssh: port 22 and for https: 443 and so on.
- You can also open up ranges.



Final word on security groups

- **Most** of the time when users can't connect to resources in your VPC the **first place** you should check is the security groups within the VPC.
- Remember that to ssh in you have to have port 22 open. ***This is not done by default*** so make sure that you have opened up the correct port in the security group.
- ALSO- kill security groups you're not using- they spin out of control quickly.



VPC Final word

- The foundation of all of your stacks **START** with the VPC. It is *strongly recommended* that you build each stack within it's own VPC **BUT...**
- Another thing to note: you are limited to **five VPCs per region** on AWS (by default- you can ask for more) so if you need more...bring your stack to a different region.



Destroy your VPC

- Now let's destroy our VPC:
 - Go to the aws console, select the vpc that we just created in the checkbox on the left, go to actions, and select **Delete VPC** from the drop down.
- We need to do this because, again: limit of five VPCs per region for AWS. These VPCs are a valuable commodity!





S3 Buckets



Data foundations for applications

- So another big part of creating a large scale applications on AWS is the reliability of your data.
- MYSQL instances crash. Postgres servers become unreachable. Data gets corrupted. How do we ensure that our data remains reachable in these events?
- Essentially the recommended way, in AWS, is to create a **data lake** model which means utilizing S3 buckets to store your data (usually as zipped files).



Managing data in AWS

- When approaching the creation of a (mostly) serverless application on AWS there is only one piece of it that will not be ephemeral: the data.
- Data should be the only **permanent** part of your application (yes- including the database). The idea here is that the raw data produced and consumed by your application lives somewhere where it will NOT disappear with the **rest of the code** when you put stuff up/take stuff down.



Data Philosophy

- So this is an important philosophical idea around architecting in AWS (so important that it's worth repeating): **nothing in the infrastructure should be permanent here except for the data.**
- So considering this- our data had **better be** robust, accessible, and in a form that meshes with just about any database in the AWS world.
- The answer to this is to get everything we can down into .txt files and zip them...which means we need a place to store those text files...



S3 Buckets

- So once our VPC is created and we have a network that we can use to connect together all of our various services the next step is to set up our data layer
- S3 buckets provide a fantastic way to save our data in a single abstraction that can be utilized by pretty much any of AWS' data repositories (DynamoDB, Postgres, MYSQL, Redshift, etc)



What are s3 buckets?

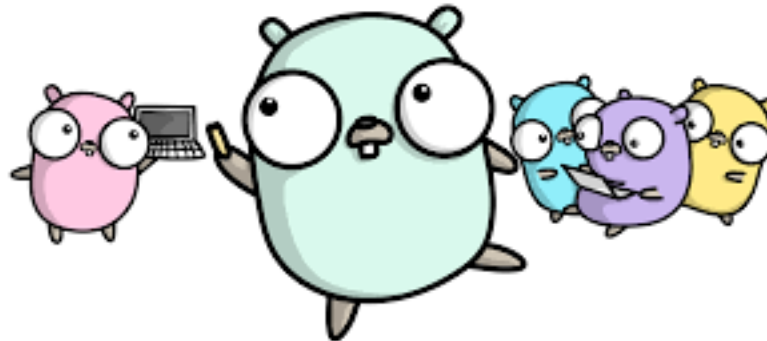
- S3 buckets are basically just directories that can store data. We can put in filesystems that can direct us to where our data files are stored.
- The idea here is that the s3 bucket will be a storage repository for unstructured data. It isn't intended that we directly query s3 (though we can)....instead it is the data source for moving things into Redshift, Dynamo, Kinesis, etc.



S3 filesystem structure for data

- So the ideal way for us to get data into s3 buckets is to set up a filesystem with dates (this is the traditional way to do it)
- You can do this by the hour, by the minute, or by the second (up to you)- but your filesystem should be:

```
Myfilesystem/2019/05/24/10/02/30/somedata.tar.gz
```



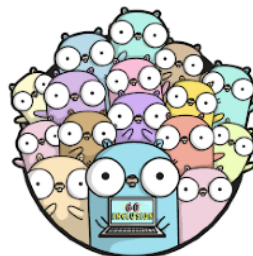
Getting Data INTO your filesystem

- So up to this point we've been addressing **how** to store our data but we haven't talked a lot about the specifics around how we will move data around.
- The other advantage of using s3 for this type of data storage is that there are numerous options for getting data into s3.
 - We can **mount** the data from another folder (in AWS speak this is a "sync")
 - We can use **lambdas** to move data in
 - We can use **Kinesis**
 - We can put backups of existing tables in there



More getting data into s3

- There are NUMEROUS methods to move data into s3 buckets that are outside of the scope but the broader takeaway here is that we want to use s3 as our **base level data store**.
- Even **within** s3 there are further options- like **s3 glacier** which can be utilized for long term storage of data that you hardly ever use.
- If you have a situation where you are getting tons of data in you can move data > 6 months old into glacier for storage.



Levels of data

- Let's create another distinction here in how most applications utilize data...let's call it **cached** data- which the data that is *actively used* by the application and the **stored data**- which is the data used in the SQL database.
- The **stored data** is the data that will be used by your analysts, data scientists, and Business Intelligence groups to do analysis. It's stuff that isn't **actively** being used by your application.



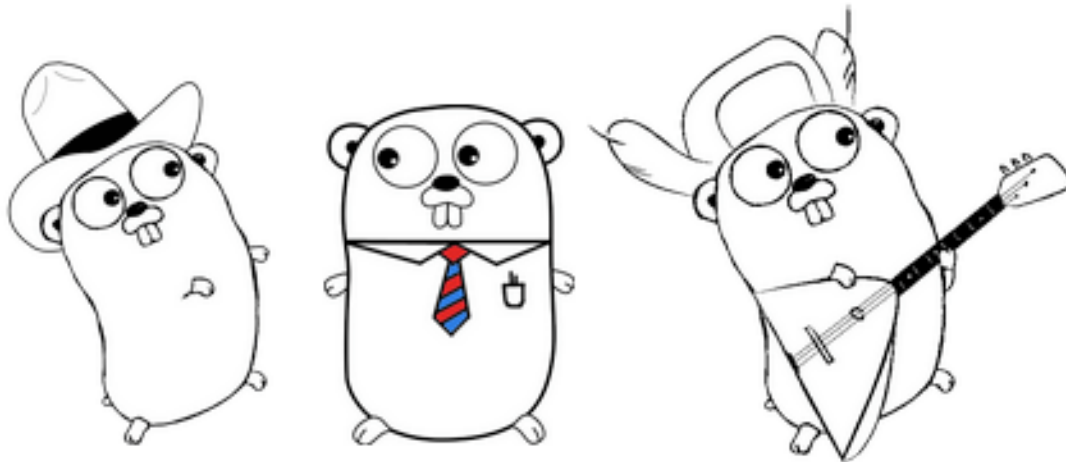
Data Levels (continued)

- The temporary or ***cached*** data is the data that your application is using. Things like “current score” for a video game or “current level” would be cached data while “layout of the level” would be ***stored*** data.
- ***Stored Data:*** is the type of data that we can set up a TIMER to send to the s3 bucket (every hour, every day, every 15 minutes, or all of the above). It’s usually structured to make it easier for analysis.
- ***Cached data:*** is usually unstructured and needed within milliseconds.



Putting data in S3

- With RDS instances in AWS we have the option to back up all data to S3 buckets...which we will do in the lab. Using AWS resources will **generally** allow you to do this without having to manually add in a “move data once an hour” command.
- Restoring backups from s3 is another easy move to make from AWS. We'll do a demo on that next.



A glowing lightbulb with a warm orange and yellow light emanating from it. The filament is visible and glowing. The word "Elasticache" is written in white, bold, sans-serif font across the center of the bulb.

Elasticache



But what about data I'm using?

- So for data being used by the application it is generally expected that you will add in a **caching layer** that will give you the ability to store your data in binary form and retrieve it quickly.
- The good news is that AWS comes with a lot of options for this...the most popular being the **elasticsearch** service.



Confused????

- GOOD! Ask Questions!!

