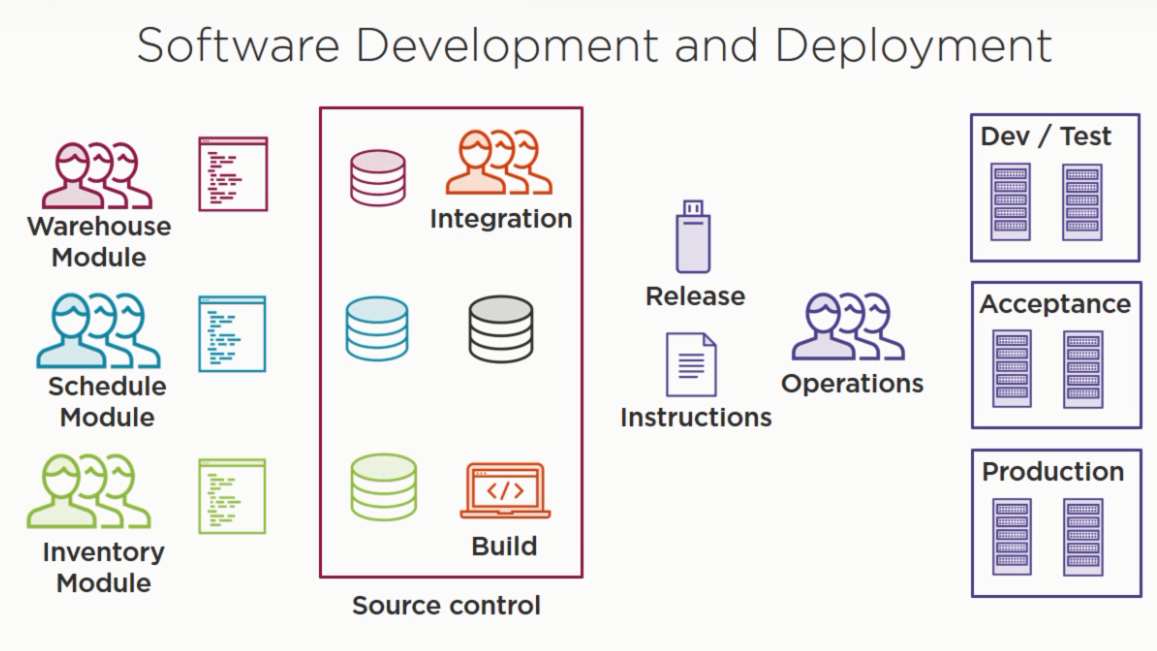
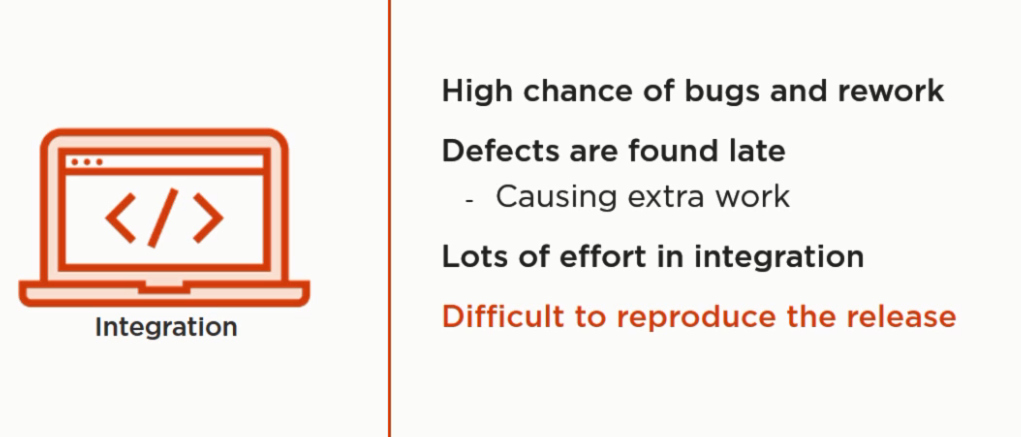
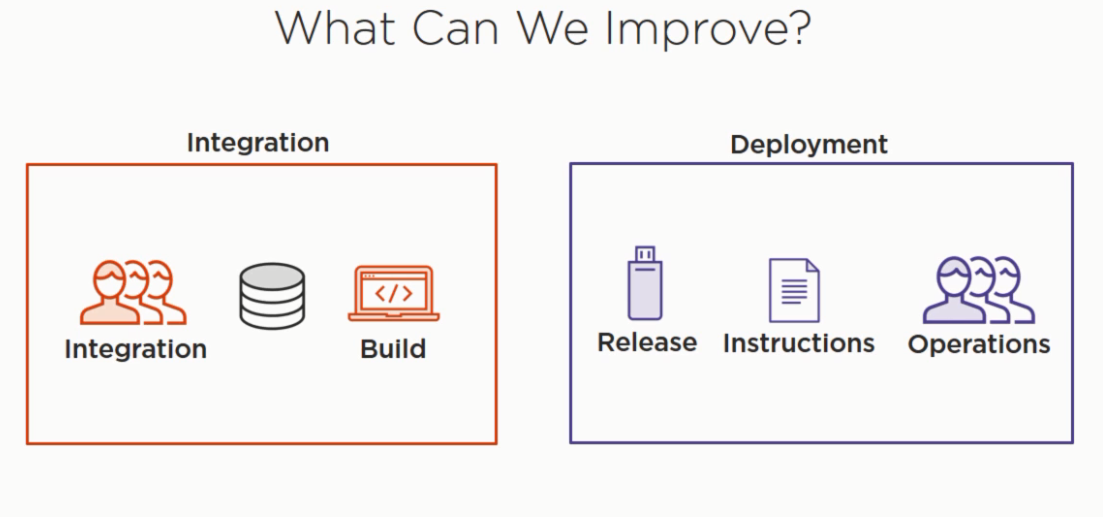
**Continous Integration and Continous Deployment**



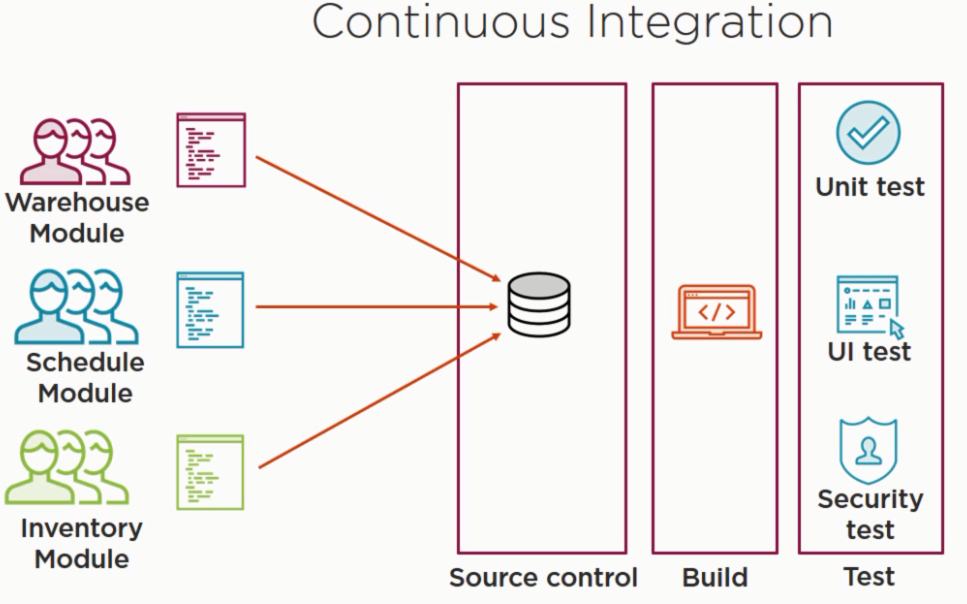
The software teams make the code on the different branches and the integration team combines all branches into one branch every iteration and then build the code in their own system. In this build process, the individuals get the different result as per the configuration of the system. Then the integration team forward the resulting release and installation instructions to the operations teams. The operation team further takes the release and install it on the different environments

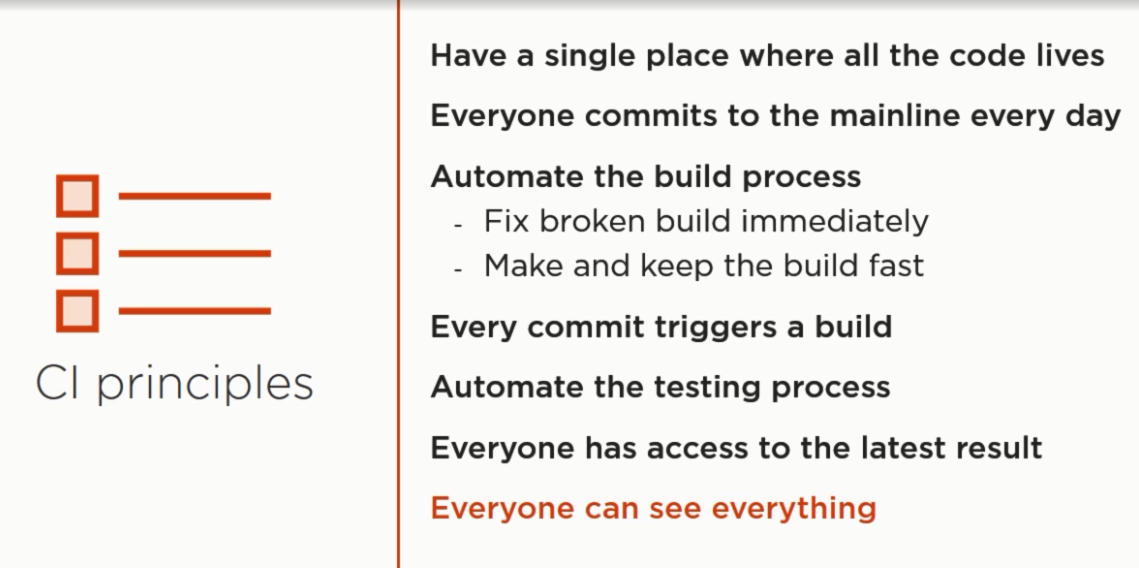
The main problem

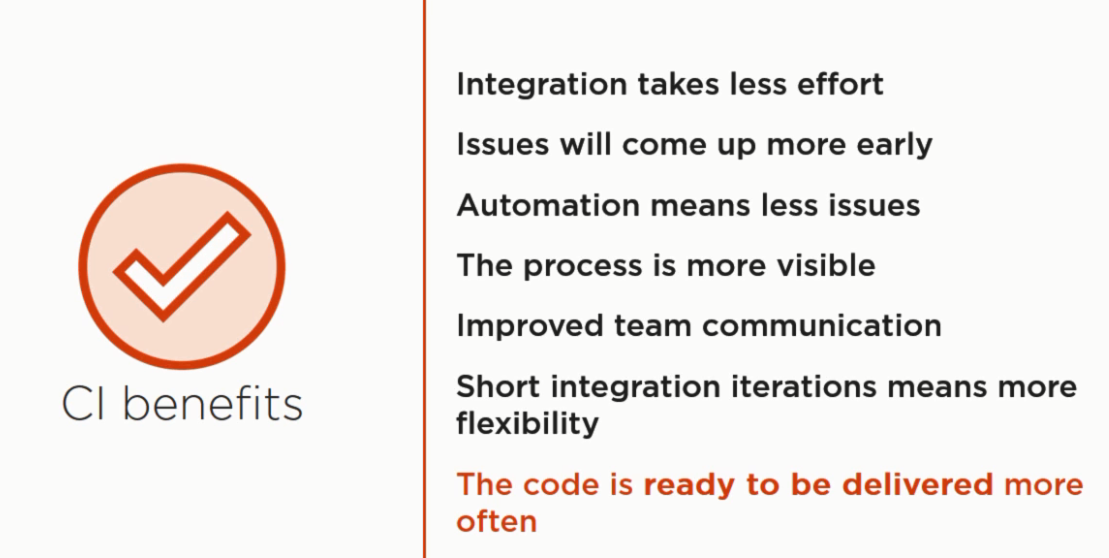












**What can CI accomplish?**

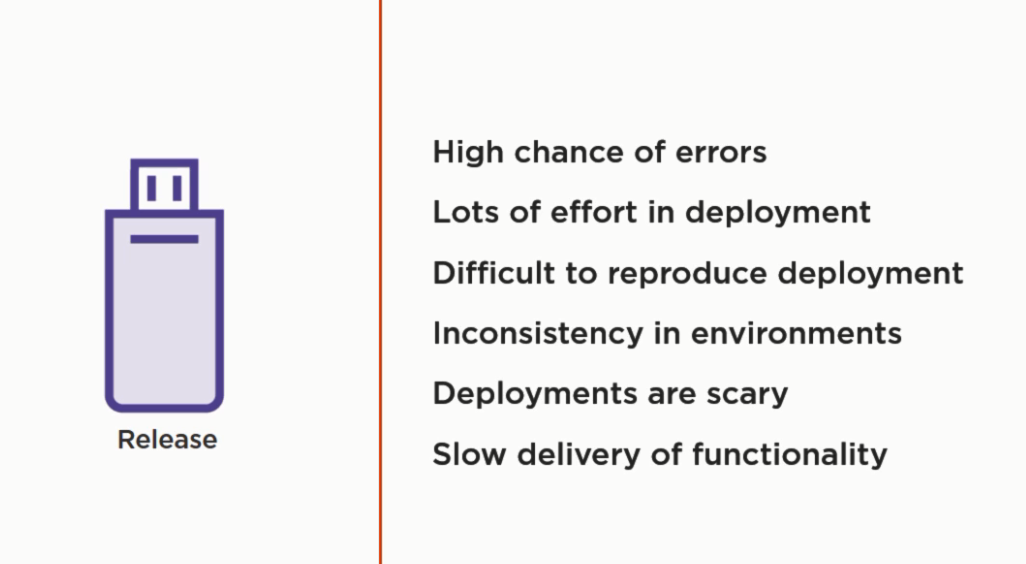
* **Higher Quality**
* **Faster Delivery**
* **Lower Costs**
* **More Flexibility**



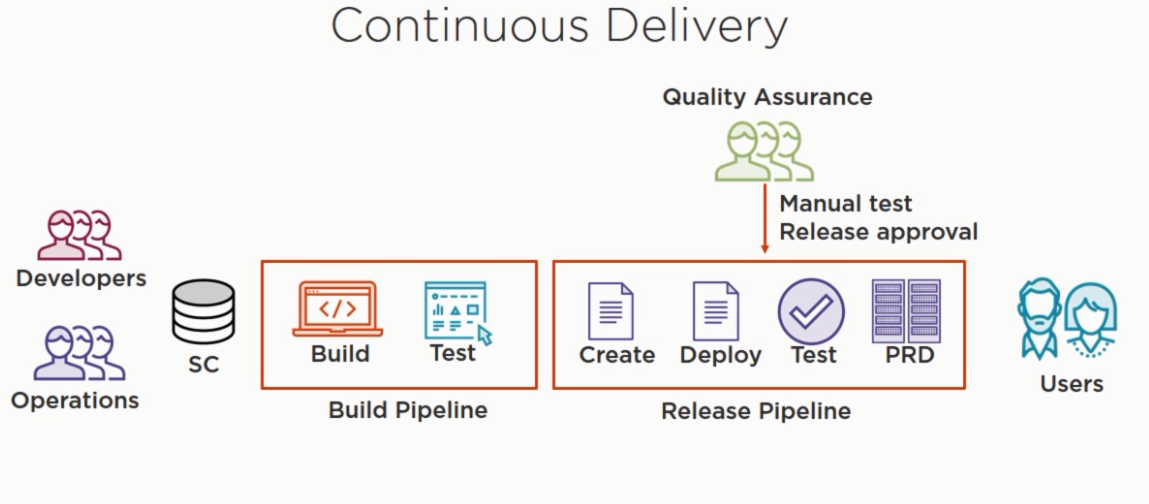
**Continuous delivery**

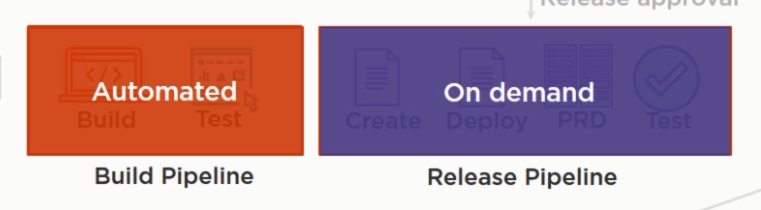


Problems



**CD**





**Terraform**

**Write, plan, create infrastructure as a Code!**

**Automating infrastructure**

HashiCorp Terraform enables you to safely and predictably create, change, and improve infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned.

**Components**

Terraform executable, we can add it to path variable and invoke terraform from the command line

One or more file (Terraform file) make up the desired deployment, other benefit is that we can extract and reuse some of the components and use it for the future deployments

Terraform state file – preferences

Example if some is using AWS to host the infrastructure and we need access key (credentials), we should not directly store the credentials in the source code due to security concerns.

However, terraform provides us the ability to store these kind of credentials using variables

**AWS – provider in terraform**

Credentials need to be defined inside the provider. To see all the properties of the particular provider, we can see the documentation that to know various components a particular provider supports.



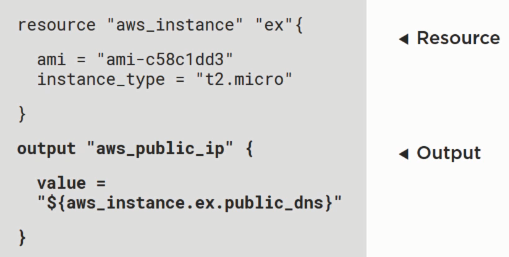
AWS – server to host the web and database components

That is called a **resource** in terraform terminology

The resource contains several arguments that can be either hardcoded or passed using the variables. For example, defining the instance type in this case t2.micro, maybe for production we need to use t2.medium or any other c4.extralarge

**Output** – in this we take the public id and extract the DNS information from it. We can pass it to something else or maybe check if the website is deployed properly or not.

There are some other sources as well that can make up the terraform file such as data sources, provisioners and modules



**Terraform scripts – configuration files.**

**The file has an extension of .tf or .tf.json if that’s a JSON configured file. There are resources and modules in the configuration files.**

A resource describes a single object file and a module describes a set of resources and their necessary relationship between them, that creates a larger unit of configuration

**Define variables**

Variable “aws\_access\_key”{}

Variable “aws\_secret\_key”{}

Variable “private\_key\_path”{}

Variable “key\_name”{}

Default = “PluralSightKeys”

}

**Define provider**

Provider “aws”{

Access\_key = “${var.aws\_access\_key}”

Access\_key = “${var.aws\_secret\_key}”

Region = “us-east-1”

**Define Resources**

Resources “aws\_instance” “nginx”

Ami =”ami c58c1dd3” - amazon services machine image (using the amazon inbuilt linux image)

Instance\_type = “t2.micro ”

Key\_name =”${var.key\_name}”

(note: the private key the user have and the public key the amazon have can be combined to make this key\_name, using this key name we can SSH into the amazon instance )

Connection {

User = “ec2-user” //default user for amazon ec2

Private\_key = “${file(var.private\_key\_path)}”

}

Provisoner “remote exec”

Inline = [

“sudo yum install nginx -y”,

“sudo service nginx start” // we can also pass a script here. Instead of 100 lines of inline code

]

**Define output**

output "aws\_instance\_public\_dns"{

value ="${aws\_instance.nginx.public\_dns}"

}

We need to create an **Amazon Web Services account to use EC2** (elastic compute cloud) and also the **access keys** are generated to be put into the script



Refer this git for examples

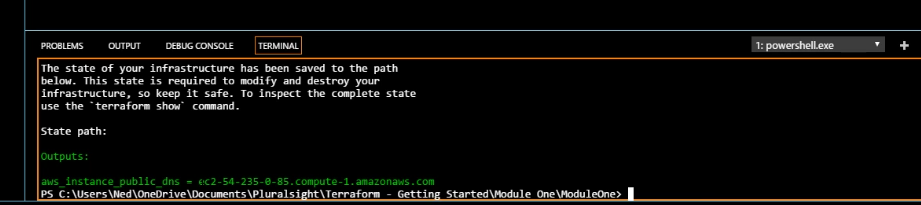
<https://github.com/ravsau/aws-labs/tree/master/terraform-aws>

**We can use the following command to plan**

terraform plan – var-file=’..\terraform.tfvars’

**To apply we can just replace the plan command with apply**

terraform apply – var-file=’..\terraform.tfvars’



**Updating your configuration with more resources**

**Terraform state and update**

**Planning updates**, see what terraform is going to do and plan updates

Terraform state file is a JSON format file which is human readable but barely needs to be changed.

In this file, we have **resource mapping and metadata** so it takes the configuration and multiple configuration files and **maps into the resource tree** that allows it to orchestrate across multiple providers by building a **resource tree and a dependency tree** between the various resources and it also stores the meta data of the resource for example last known good state, computed information submitted to provider, ex DNS address of the load balancer – this information is not known ahead of time but it Is generated by the AWS and then it takes the information from load balancer

**Locking** - when deploying a new instance or making updates to the instance that state file gets locked. It is important when you have multiple people working on infrastructure and you have stored the state file remotely as opposed to the local location, then you can guarantee that when you are making changes to the infrastructure, then everybody else gets locked out.

**Storage** - File can be stored locally by default, when you run terraform apply it creates a state file in the same folder that your terraform files are in and can be stored remotely that could be nfs share on the local network or it could be some sort of hosted application hashicorp’s console, which is an hosted service that can hold that state or you can put your state in an S3 bucket, as long as all the people can access the state file, update and lock it, then it doesn’t matter where you store your state file

Environments

**The scenario**

**Data type and security groups**