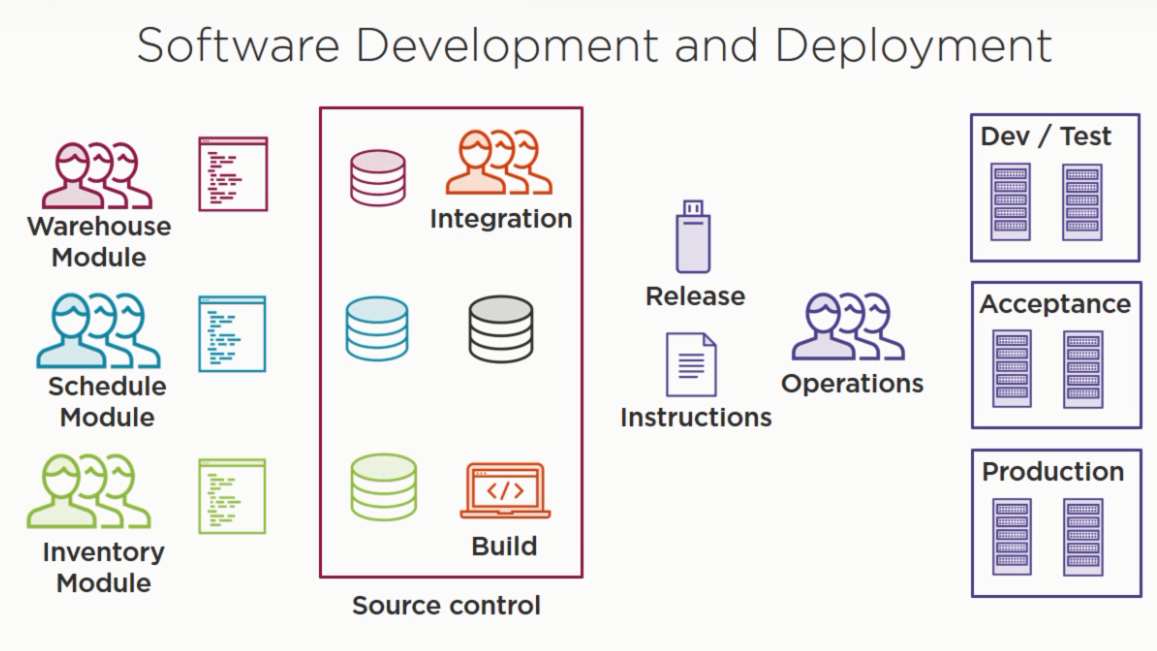
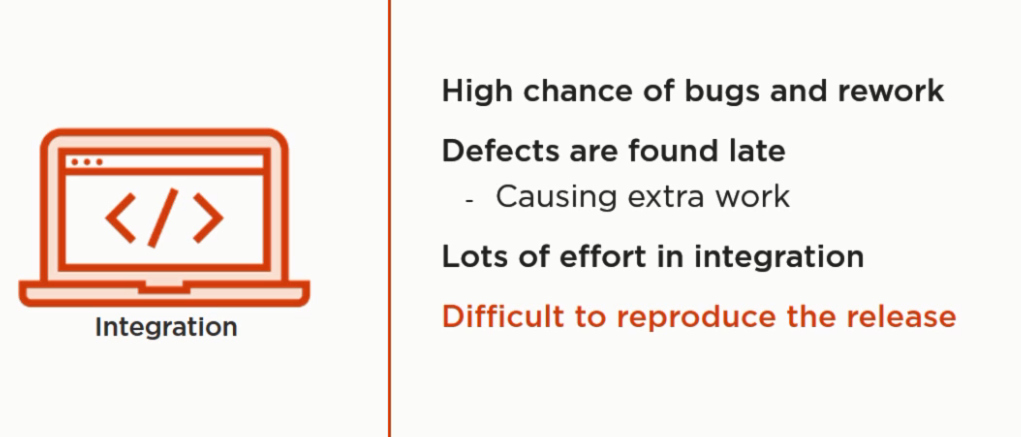
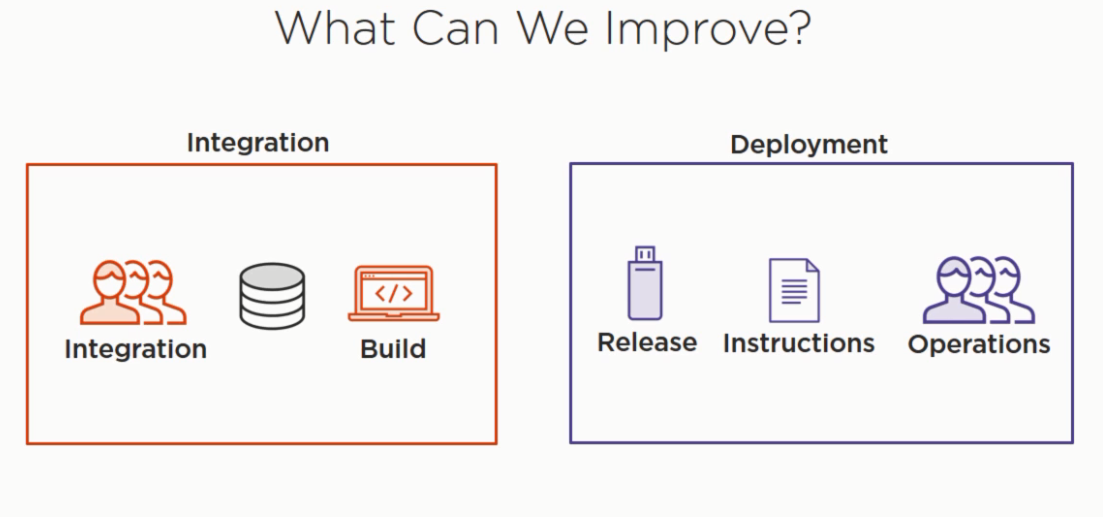
**Continous Integration and Continous Deployment**



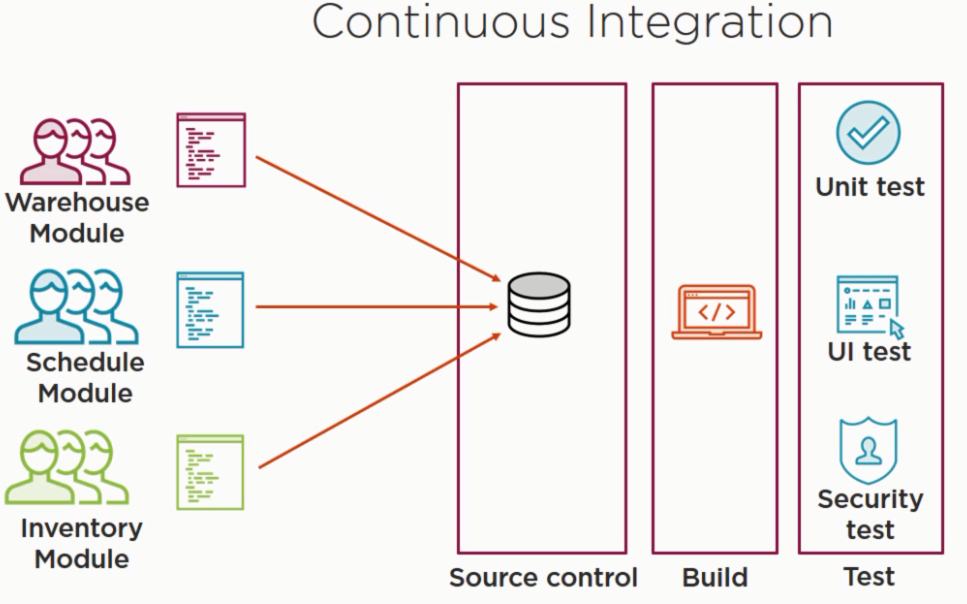
The software teams make the code on the different branches and the integration team combines all branches into one branch every iteration and then build the code in their own system. In this build process, the individuals get the different result as per the configuration of the system. Then the integration 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

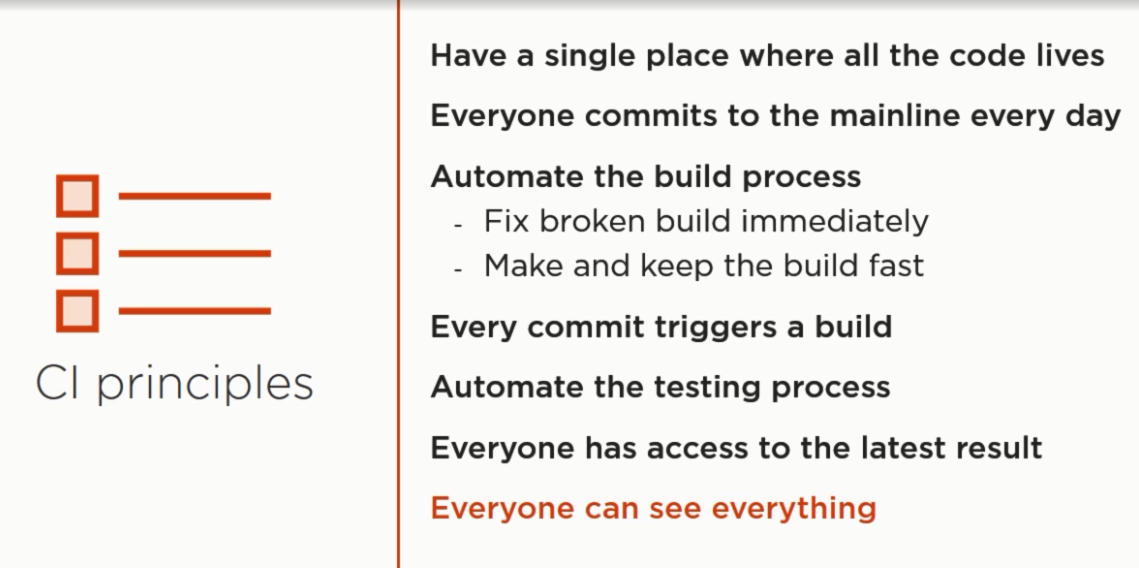
The main problem

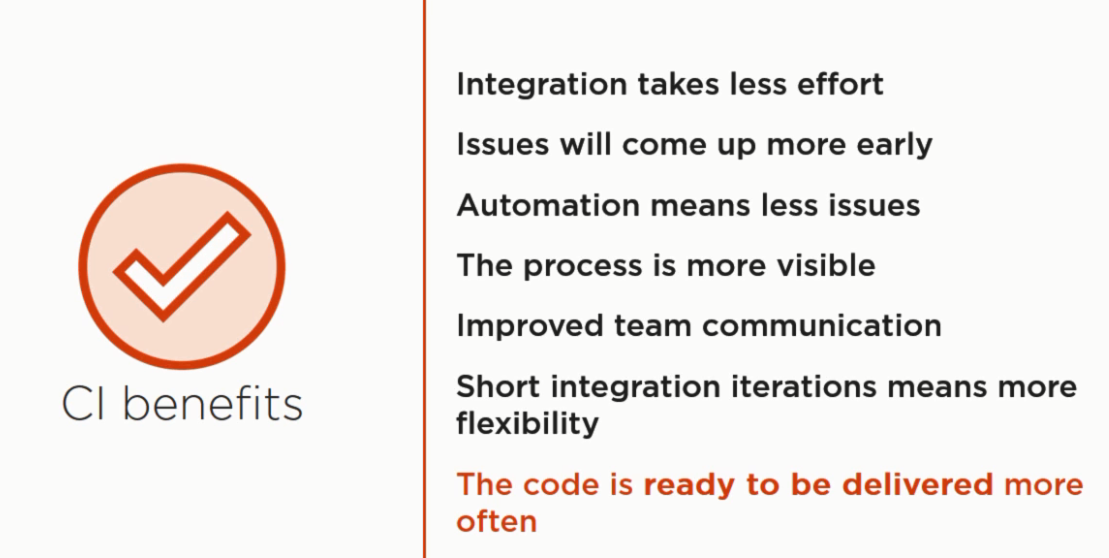












**What can CI accomplish?**

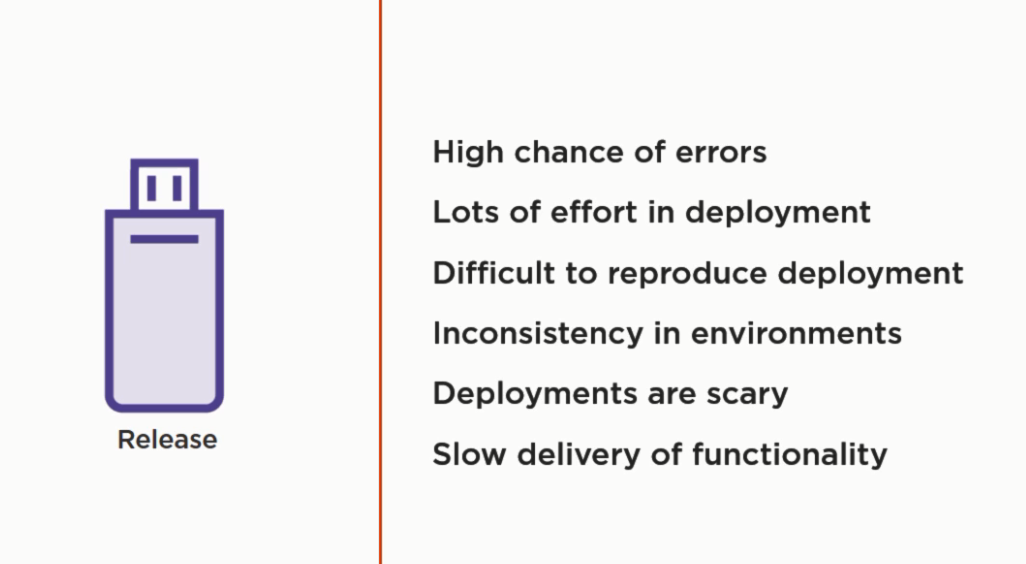
* Higher Quality
* Faster Delivery
* Lower Costs
* More Flexibility



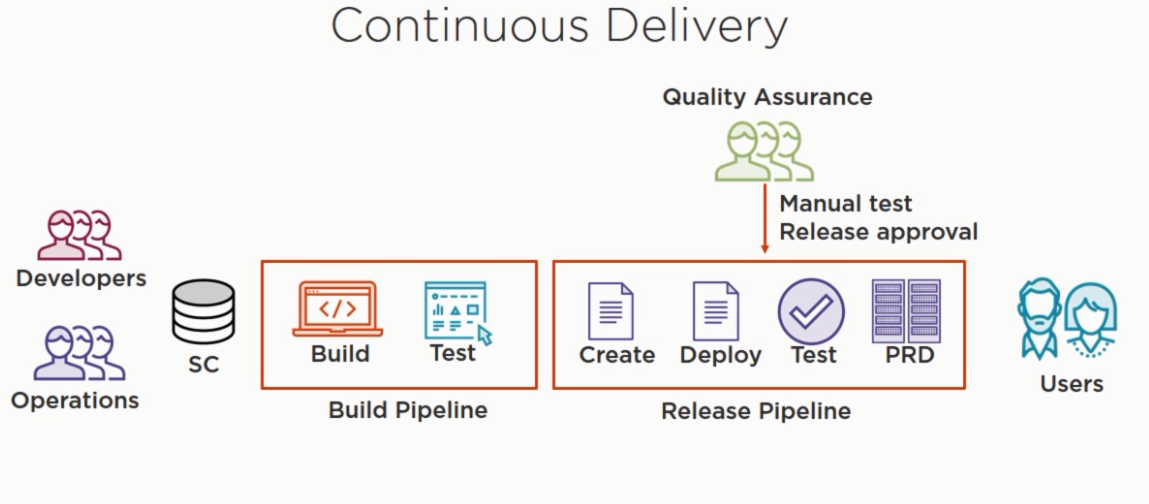
**Continuous delivery**

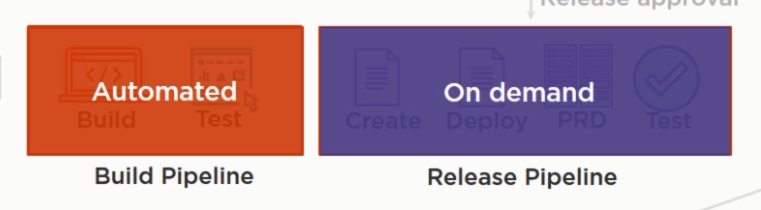


Problems



**CD**





Cloud and distributed systems

Configuration management automation tools: Ansible, Chef

Container based deployment – Docker, Kubernetes, Packer

Enterprise Server Management

SCM or version control tools – Git Hub

Develop automation and Infrastructure as a code to improve the operational consistency and

**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.



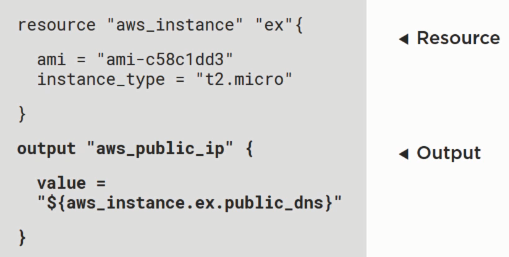
AWS – server to host the web and database components

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”

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

}

Provisoner “remote exec”

Inline = [

“sudo yum install nginx -y”,

“sudo service nginx start”

]

**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>