Release Pipelines in Microsoft Ecosystems

Warren Frame, Harvard University

Michael Greene, Microsoft

usenix LISA16

December 4–9, 2016 | Boston, MA
www.usenix.org/lisa16 #lisa16

whoami

- Warren Frame
  - Research Computing at Harvard University
  - @pscookiemonster
  - Ramblingcookiemonster
  - wframe
- Michael Greene
  - Enterprise Cloud Engineering CAT Team at Microsoft
  - @migreene
  - mgreenegit
  - migreene

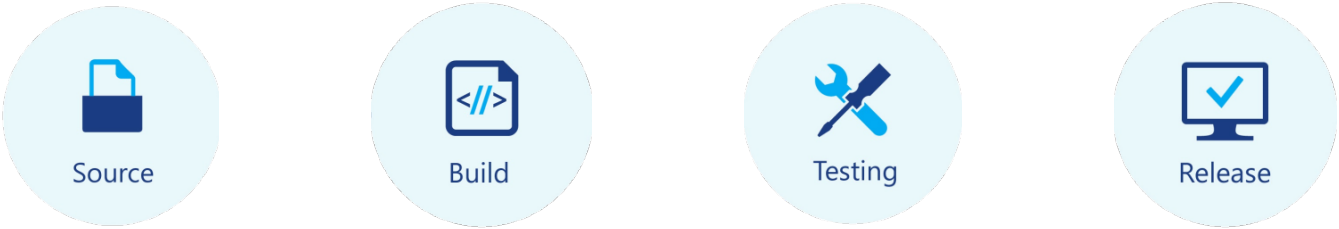
Stuff

- Slides
- Demos!
- Slides at bit.ly/lisa16pipeline
- Cleanup,

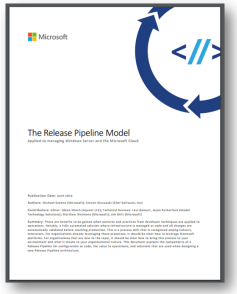
Configuration as Code

- Everything-as-a-service, APIs galore
- Living documentation
- Abstract out complexity. Scripts -> Modules -> DSC -> key:value
- PowerShell DSC is a platform that all solutions can use to deploy and manage Windows Server
- Azure Resource Manager templates
- You still need to know the underlying systems you will manage
- Release pipelines can bring sanity and consistency to managing this

Release Pipeline

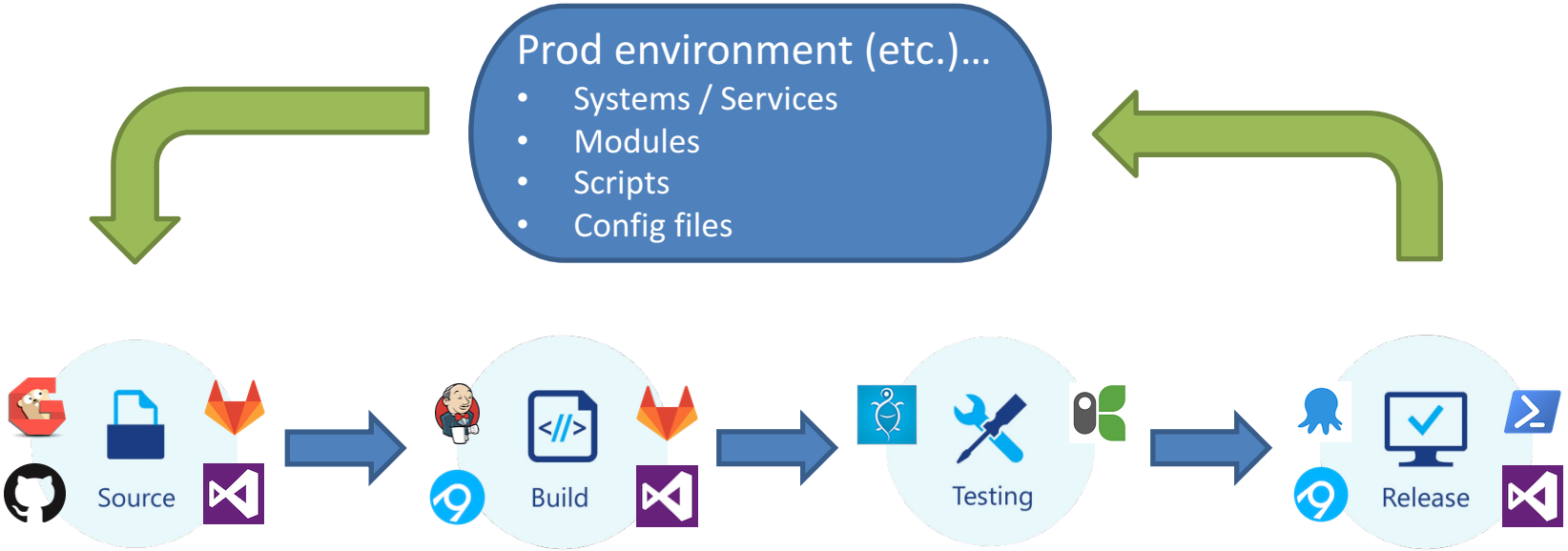


Why?



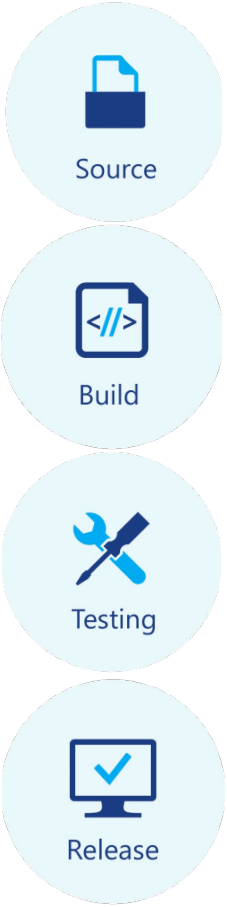
aka.ms/trpm

Release Pipelines



Example Workflow

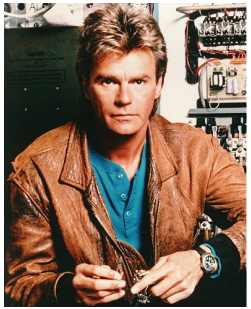
- Make a change, push to source control*
- Build system does the rest. For example:
  - Run tests against your code
  - Spin up test services/infrastructure for more tests
  - Build artifacts (packages, configs, etc.)
  - Deploy things (artifacts, systems, services, etc.)

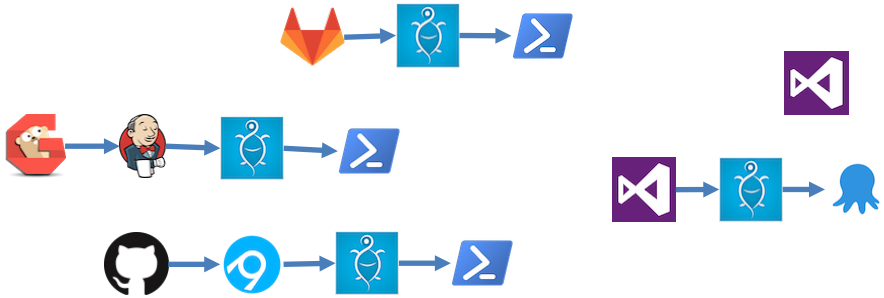


* You might run through source-build-test loops locally until happy, before pushing

Tooling

"a bunch of random open source projects bound together with duct tape and chewing gum"





Tools: Source

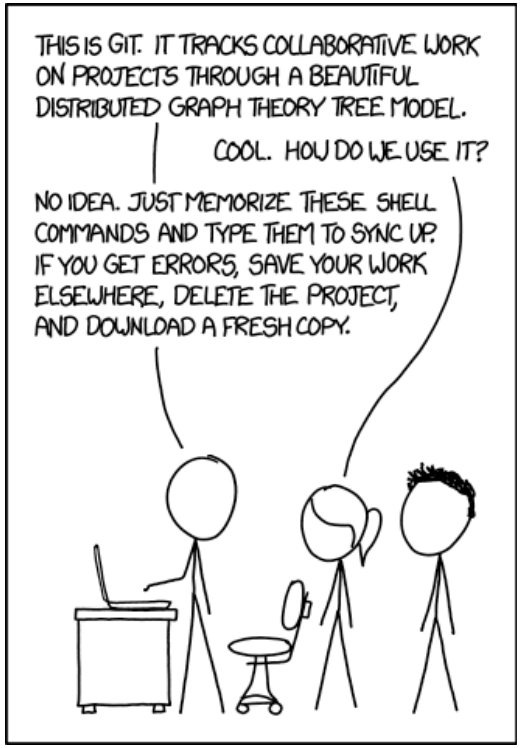
Git? Mercurial? ~~SVN?~~

CLI:

- Git for Windows
- PoshGit

GUI:

- GitHub Desktop
- Atlassian SourceTree
- Many others



Demo: Source

Git

Visual Studio Code

Tools: Build Systems

- Jenkins, GitLab CI, VSTS, etc.
- Prefer build-as-code
  - e.g. [Jenkinsfile](), [appveyor.yml](), [.gitlab-ci.yml]()

Xainey init commit

1 contributor

20 lines (16 sloc) | 589 Bytes

```
node('windows') {
    // May not need this stage if using Jenkins SCM to checkout Jenkinsfile
    stage 'Stage 1: Build'
    git url: 'https://github.com/Xainey/DSCTextfile.git'

    stage 'Stage 2: Analyze'
    posh './build.ps1 -Task JenkinsAnalyze'

    stage 'Stage 3: Test'
    posh './build.ps1 -Task JenkinsTest'

    stage 'Stage 4: Approve Publish Module'
    input 'Deploy to Module Respository?'

    stage 'Stage 5: Publish Module'
    posh './build.ps1 -Task JenkinsDeploy'
}
def posh(cmd) {
  bat 'powershell.exe -NoProfile -ExecutionPolicy Bypass -Command "& ' + cmd + '"'
}
```

Tools: Build Automation

- Invoke-Build, psake
- Similar to rake, make, bake, cake, grunt, gulp, msbuild, etc.

```
$SomeSharedVariable = 'or not'

# Entry points.  Dot is default.
Task . Deploy

Task Deploy Init, Build, Test, Deployment

Task Init {
    "Initialize things..."
}

Task Build {
    "Build a thing... $SomeSharedVariable"
}

Task Test Init, Build, {
    "Assert a thing..."
}

Task Deployment {
    "Deploy a thing..."
}
```

```
PS E:\Olisa> Invoke-Build
Build . E:\Olisa\.build.ps1
Task /./Deploy/Init
Initialize things...
Done /./Deploy/Init 00:00:00.2157838
Task /./Deploy/Build
Build a thing... or not
Done /./Deploy/Build 00:00:00.0320375
Task /./Deploy/Test
Assert a thing...
Done /./Deploy/Test 00:00:00.0350480
Task /./Deploy/Deployment
Deploy a thing...
Done /./Deploy/Deployment 00:00:00.04957
Done /./Deploy 00:00:00.4143803
Done /. 00:00:00.4491511
Build succeeded. 6 tasks, 0 errors, 0 wa
```

```
PS E:\Olisa> Invoke-Build -Task Build
Build Build E:\Olisa\.build.ps1
Task /Build
Build a thing... or not
Done /Build 00:00:00.0300794
Build succeeded. 1 tasks, 0 errors, 0 warnings

PS E:\Olisa> Invoke-Build -Task Test
Build Test E:\Olisa\.build.ps1
Task /Test/Init
Initialize things...
Done /Test/Init 00:00:00.0387023
Task /Test/Build
Build a thing... or not
Done /Test/Build 00:00:00.0292894
Task /Test
Assert a thing...
Done /Test 00:00:00.1388122
Build succeeded. 3 tasks, 0 errors, 0 warnings
```

Demo: Build

TFS 2017

psake (build automation)

github.com/powershell/demo_ci

Tools: Testing

- Pester: Test framework
- poshspec: infrastructure testing
- OVF: Operation-Validation-Framework - simplify organizing, execution, and sharing of tests.

```
Describe 'Math' {
    It 'Should add up' {
        2 + 2 | Should be 4
    }
}
```

```
Describing Math
 [+] Should add up 32ms
```

```
Describe 'Http' {
    TcpPort github.com 80 TcpTestSucceeded { Should Be $true }
    Http https://github.com RawContent { Should Match 'Search GitHub' }
}

Describe 'Hotfix' {
    Hotfix KB3199209 { Should Not BeNullOrEmpty}
    Hotfix KB3199208 { Should BeNullOrEmpty}
}
```

```
Describing Http
 [+] TcpPort property 'TcpTestSucceeded' for 'github.com' at '80' Should Be $true 486ms
 [+] Http property 'RawContent' for 'https://github.com' Should Match 'Search GitHub' 769ms
Describing Hotfix
 [+] Hotfix 'KB3199209' Should Not BeNullOrEmpty 845ms
 [+] Hotfix 'KB3199208' Should BeNullOrEmpty 763ms
```

Demo: Test

Pester

poshspec

Tools: Release

- Octopus Deploy and VSTS
  - Many pre-canned tasks
  - Flexible
  - Pretty
  - Potentially $$
- PSDeploy
  - Some pre-canned tasks
  - Deployment as code
  - Poorly written
  - Open source
- ~~Random PowerShell code~~
  - Fun to read and maintain!

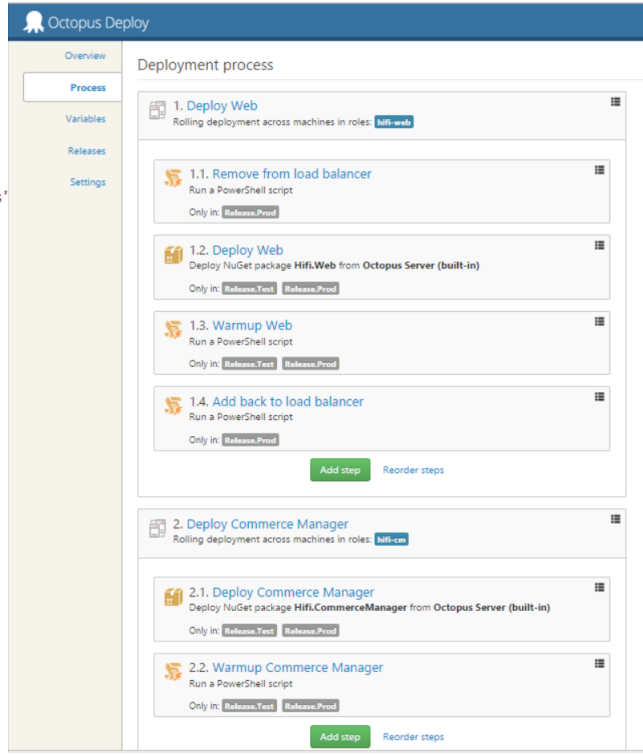
```
Deploy VSCodeExtensions {
    By PlatyPS {
        FromSource "$BHProjectPath\docs\Commands"
        To "$BHProjectPath\VSCodeExtensions\en-US"
        Tagged Help
        #...
    }

    By FileSystem {
        FromSource VSCodeExtensions
        To "$home\Documents\WindowsPowerShell\Modules\VSCodeExtensions"
        Tagged Prod, Module, Local
        WithPostScript {
            Import-Module -Name VSCodeExtensions -Force
        }
    }

    By PSGalleryModule {
        FromSource $ENV:BHProjectName
        To PSGallery
        WithOptions @{
            ApiKey = $ENV:NugetApiKey
        }
    }
}
```

```
Deploy DeveloperBuild {
    By AppVeyorModule {
        FromSource $ENV:BHProjectName
        To AppVeyor
        WithOptions @{
            Version = $ENV:BHBuildNumber
        }
    }
}

Deploy ExampleDeployment {
    By Artifactory {
        FromSource 'myscript.ps1'
        To 'http://artifactory.local:8081/artifactory'
        Tagged Prod
        # ...
```



Demo: Release

TFS 2017 - Release management

Tools: Test Harness

- Test-Kitchen
- Not just for Chef
- Roughly:
  - Run tests with a **verifier** (Pester)
  - against **platforms** (different vagrant boxes)
  - converged with a **provisioner** (dsc)
  - with the lifecycle managed by a **driver** (vagrant)
  - And test, configuration, other files copied to platforms via a **transport** (WinRM)

```
driver:
  name: azurerm # or vagrant, docker, hyper-v, openstack, etc.

driver_config:
  username: CiCdDemo
  password: 'th1s !s Awes0me'
  subscription_id: 'af168e3d-f37a-4a60-8d1e-?'
  location: 'East US'
  machine_size: 'Standard_D1'

provisioner:
  name: shell             # or DSC, Puppet, Chef, Ansible, SaltStack, etc.
  data_path: .            # Folder, copied to $ENV:Temp\Kitchen\Data
  script: start.build.ps1 # Script to invoke

verifier:
  name: pester # or shell +serverspec, shell +inspec, etc.

transport:
  name: winrm

platforms:
  - name: windows-2012r2
    driver_config:
      image_urn: MicrosoftWindowsServer:WindowsServer:2012-R2-Datacenter:latest
  - name: windows-2016
    driver_config:
      image_urn: MicrosoftWindowsServer:WindowsServer:2016-Datacenter:latest

suites:
  - name: default
  - name: anothersuite
```

```
PS > bundle exec kitchen list
Expected array default value for '--driver'; got "kitchen-v
Instance                    Driver   Provisioner  Verifier
default-windows-2012r2      Azurerm  Shell        Pester
default-windows-2016        Azurerm  Shell        Pester
anothersuite-windows-2012r2 Azurerm  Shell        Pester
anothersuite-windows-2016   Azurerm  Shell        Pester
```

Drivers

- Amazon EC2
- Azure Resource Manager
- DigitalOcean
- Docker
- Google Compute Engine
- Hyper-V
- OpenStack
- Vagrant
- vRealize Automation, Orchestrator
- vSphere

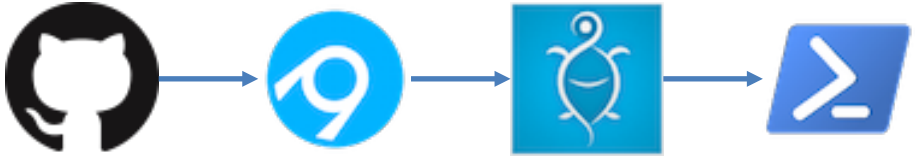
Provisioners

- Ansible
- CFEngine
- Chef Solo, Zero
- DSC
- Puppet
- Salt
- Shell

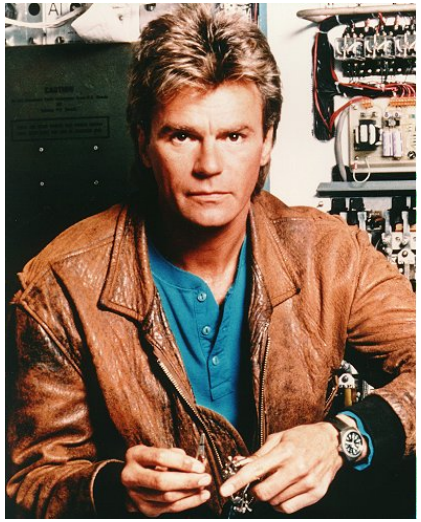
Verifiers

- Inspec
- Pester
- Shell (Bats, Serverspec, etc.)

Example Pipeline



- Source: GitHub
- Build: AppVeyor
  - Build dependencies: PSDepend
  - Build automation: Invoke-Build
  - Build helpers: BuildHelpers
- Test: Pester
- Release: PSDeploy



Demo: Example Pipeline

https://github.com/RamblingCookieMonster/lisa-kitchen-demo

What about…

- Secrets
  - ~~In source control~~
  - Built into build system?
  - Secret management – vault, passwordstate, Secret Server, credstash, etc.
- Images
  - Packer!
  - Images-as-code
  - Build images for Amazon, VirtualBox, Azure, Hyper-V(ish), etc.

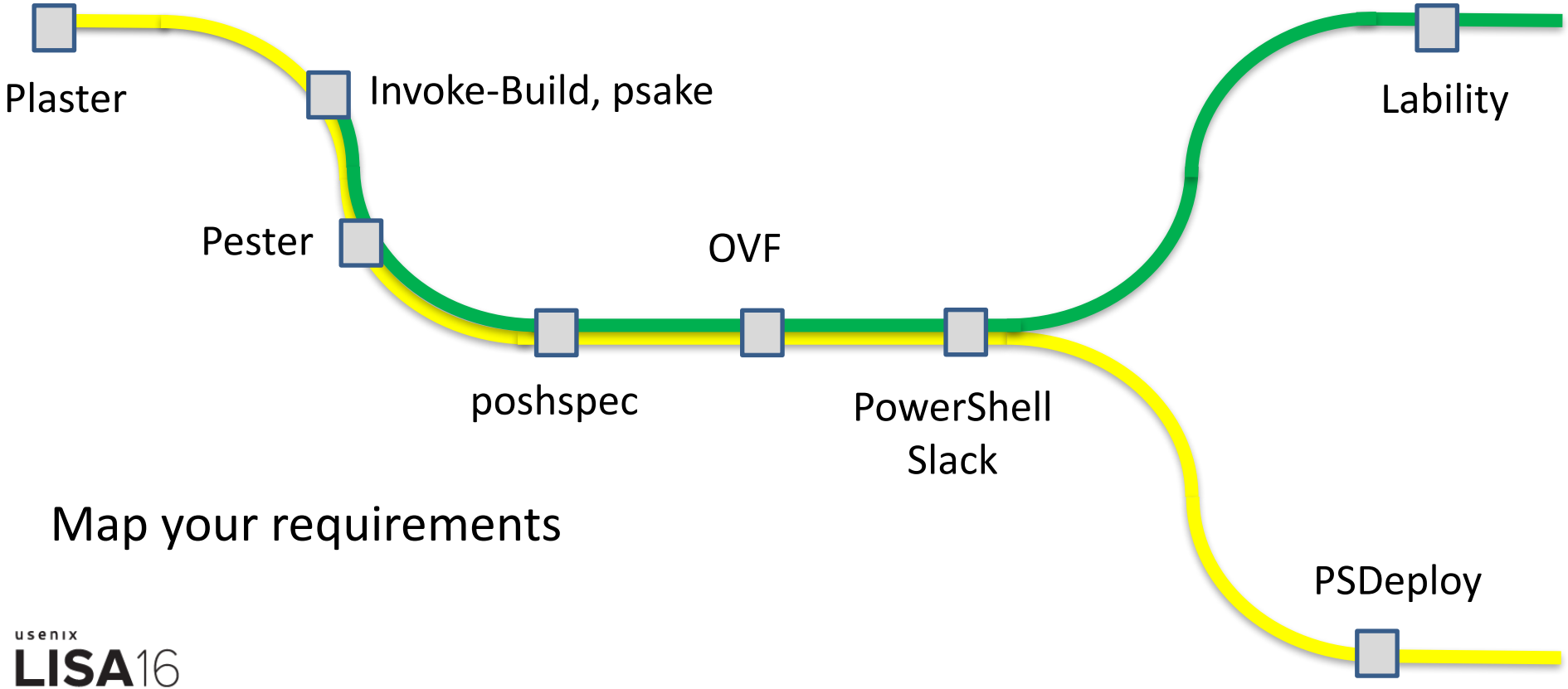
Where to start

- Source Control and/or Tests over entire pipeline at once
- Existing tools over resume-driven-development
- New service(s) / value proposition over re-engineering everything
- No luck in house? Play with GitHub+AppVeyor, VSTS, etc.

Next steps

- Open source projects could use your help!
- JIT provisioning or a dynamic pools of Windows build agents
- Windows Docker containers for testing
- Focus on ephemeral deployments over incremental changes
- Plan for day 100

Community Projects



Map your requirements

References, Diving Deeper

- The Release Pipeline Model - Michael Greene, Steven Murawski
- Building a Simple Release Pipeline in PowerShell Using psake, Pester, and PSDeploy - Brandon Olin
- Stack Overflow: How We Do Deployment - 2016 Edition - Nick Craver
- DevOps Reading List - Steven Murawski
- Reading List - Chris Hunt
- The Pester Pipeline - Chris Hunt
- Best Practices with Packer and Windows - Matt Hodgkins
- Introduction to Kitchen-DSC - Gael Colas
- Testing Ansible Roles Against Windows with Test-Kitchen - Matt Hodgkins
- Twitter, Slack, and other communities
- Etc.