Building Secure and Flexible Multi-Cloud Images with Multi-Boot Mode

A Comprehensive CI/CD Approach

Ederson Brilhante

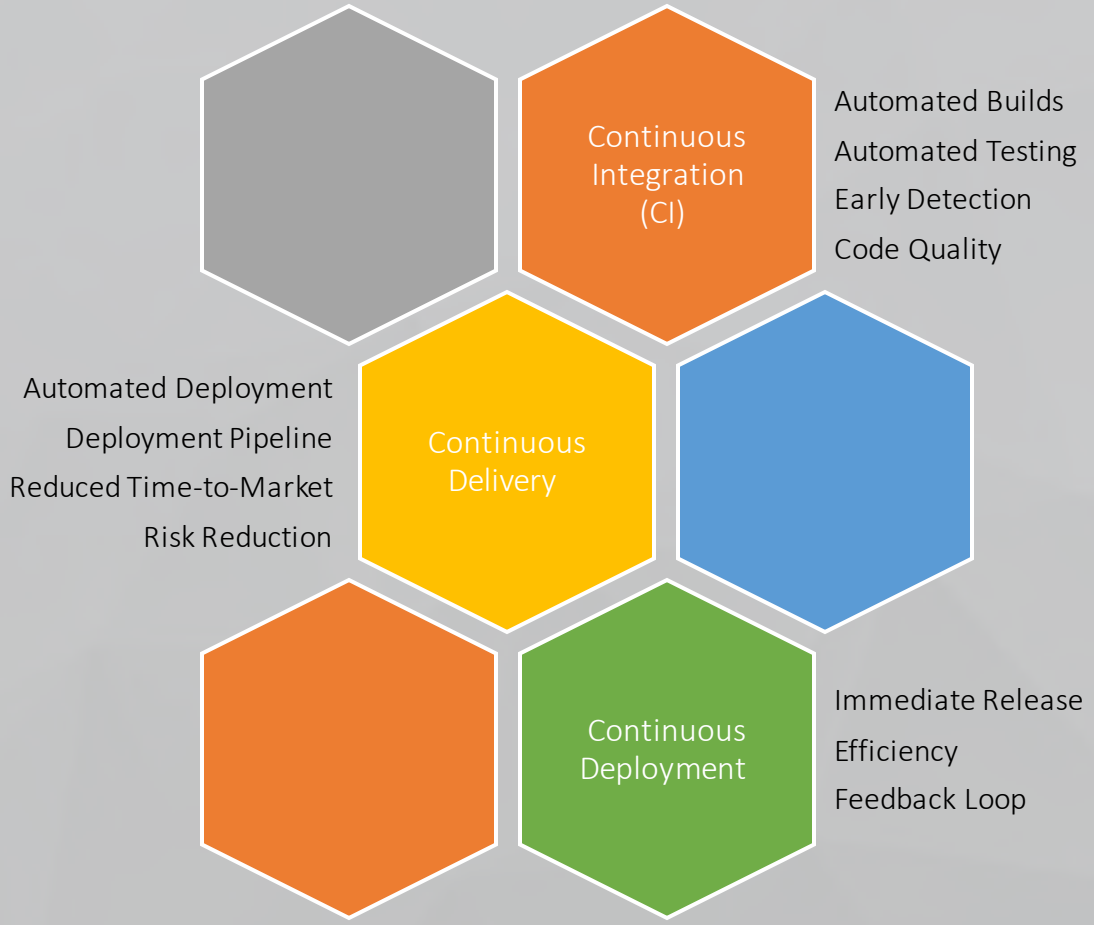
whoami

- Brazilian Expat living in Poland
- 17 years in IT
- Developer specialist with focus in Infra/Cloud

What will we see in this session?

General CICD Overview

General CICD Overview



General CICD Overview

Reliability

Frequent Updates

Fast Bug Fixes

Consistent User Experience

Security and Privacy

Responsive Features

Transparency

Availability

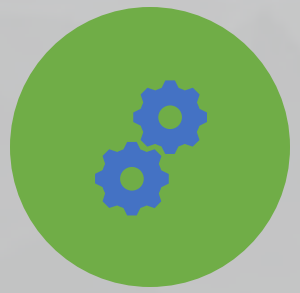
Our Context

Our Context

Multi Cloud

Multi Boot Mode

Multi-Stage Build Strategy



Code Overlap and maintenance

Our Approach

Our Approach Overview

GitHub Repo per build stage

GitHub Actions as function/components

Packer scripts to build images

Terraform modules/scripts to deploy environments for testing

Ansible Files(Build and Tests)

Regression Tests

Project Structure

Hardened Images
Builder Suite

Product Base Images
Builder Suite

Product Images
Builder Suite

Product Images
Tester Suite

Builder Suite Structure

GitHub Reusable Workflows

GitHub Composite Actions(per cloud)

Common Ansible Files

Packer Scripts (per cloud)

File with Dependency Versions

Tester Suite Structure

GitHub Reusable Workflows

GitHub Composite Actions(per cloud)

Terraform modules(per cloud)

Common Ansible Files

Why this Stack?

GitHub Actions

Empowers Developers: Efficient CI/CD pipelines

Customization: Tailored workflows for requirements

Seamless Integration: Within GitHub ecosystem

End-to-End Automation: Build, test, deploy in one place

Developer Flexibility: Easy customization and extension

Versatility: Covers various aspects of SDLC

Terraform

IaC:
Define infrastructure in code for consistency.

Multi-Cloud:
Manage infrastructure across providers.

Resource Graph:
Ensure proper provisioning order.

Modularity:
Reuse components for maintainability.

State Management:
Track infrastructure state.

Community Ecosystem:
Integrates with various services.

Packer

Speed: Quickly build images with parallelization.

Multi-Platform: Supports various platforms and formats.

Integration: Works seamlessly with configuration tools.

Security: Builds images from trusted sources.

Community: Active community and plugin support.

Immutable **Infra: Create consistent machine images.**

Ansible

Agentless: Operates over SSH for simplicity.

Declarative: YAML playbooks for readability.

Idempotence: Ensures predictable automation.

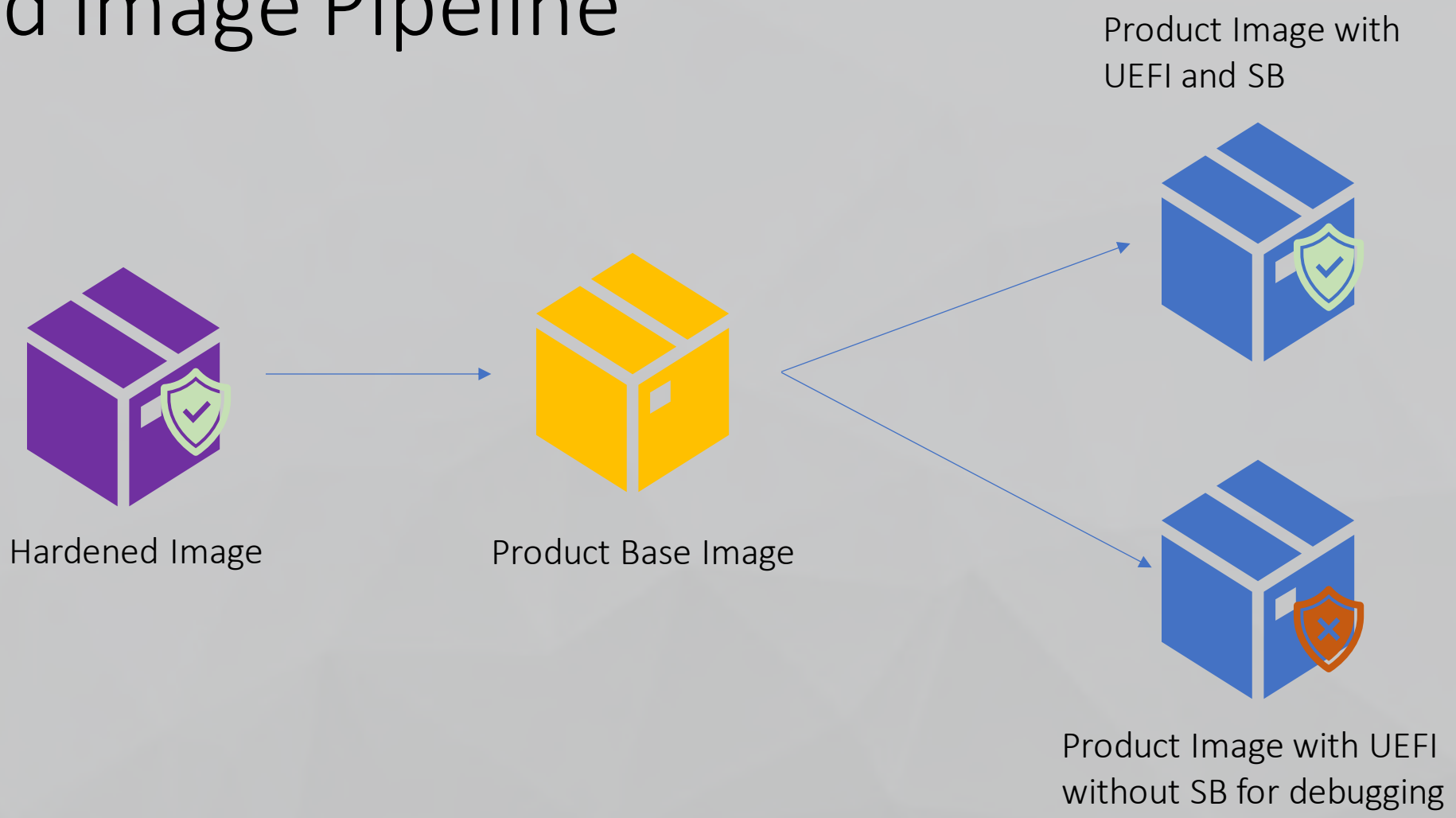
Extensible: Supports custom modules and plugins.

Community: Large community and Galaxy repository.

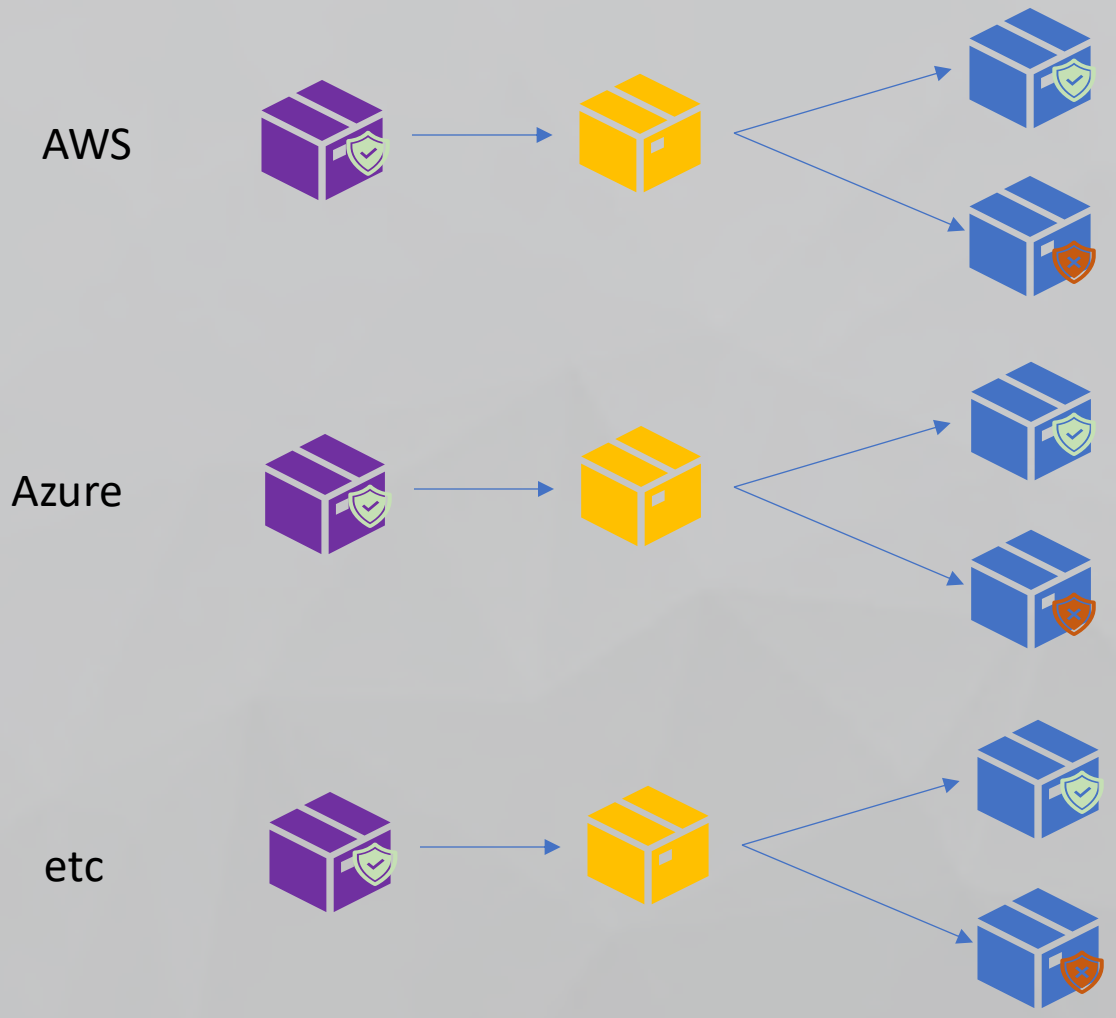
Integration: Seamlessly integrates with other tools.

Workflows

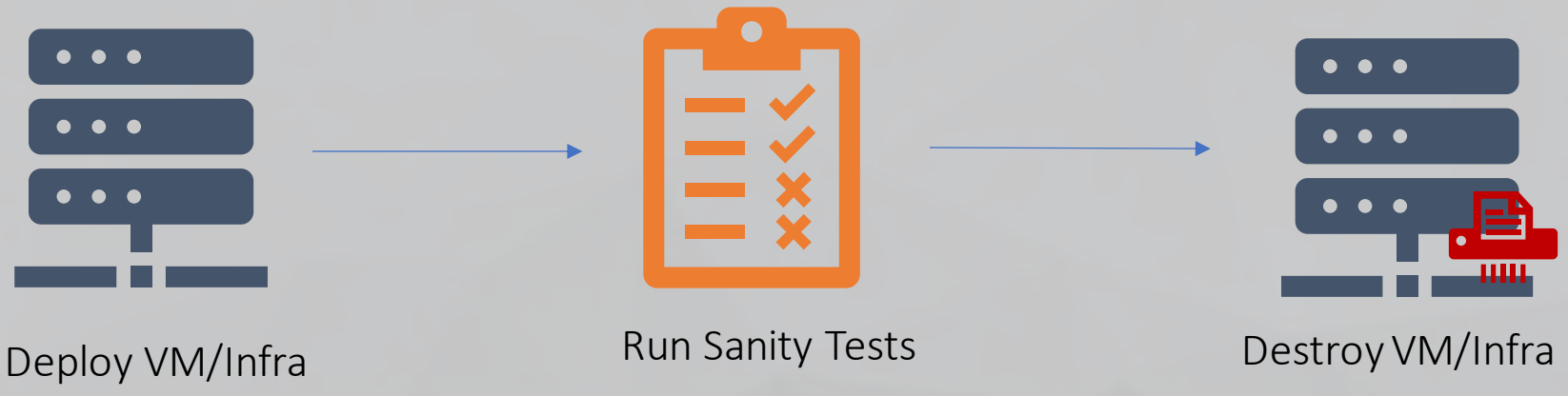
Build Image Pipeline



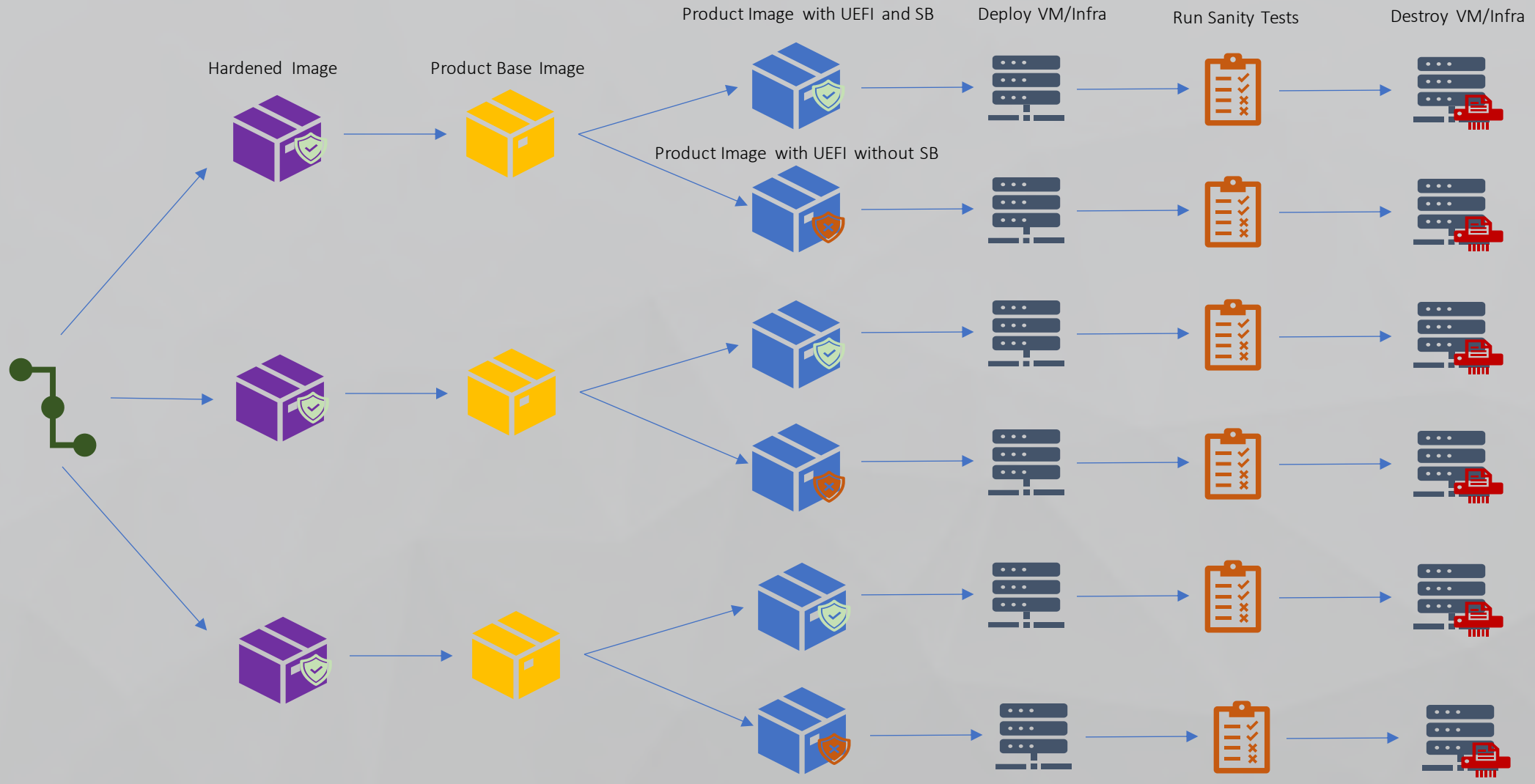
Build Image Pipeline



Test Image Pipeline

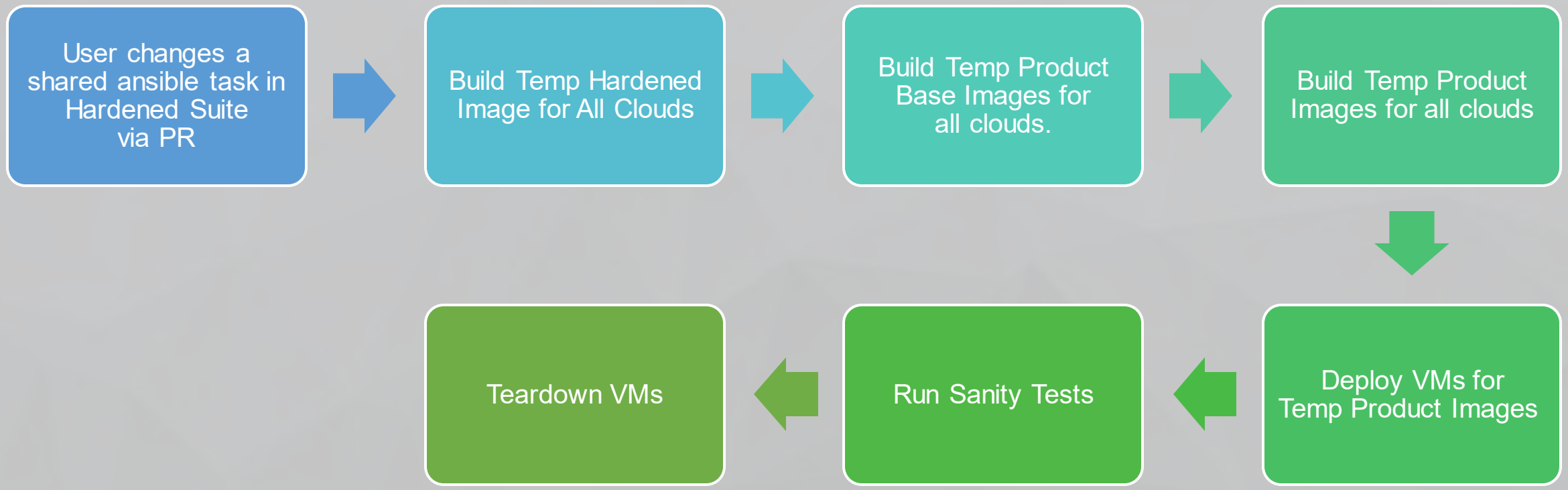


Full Pipeline



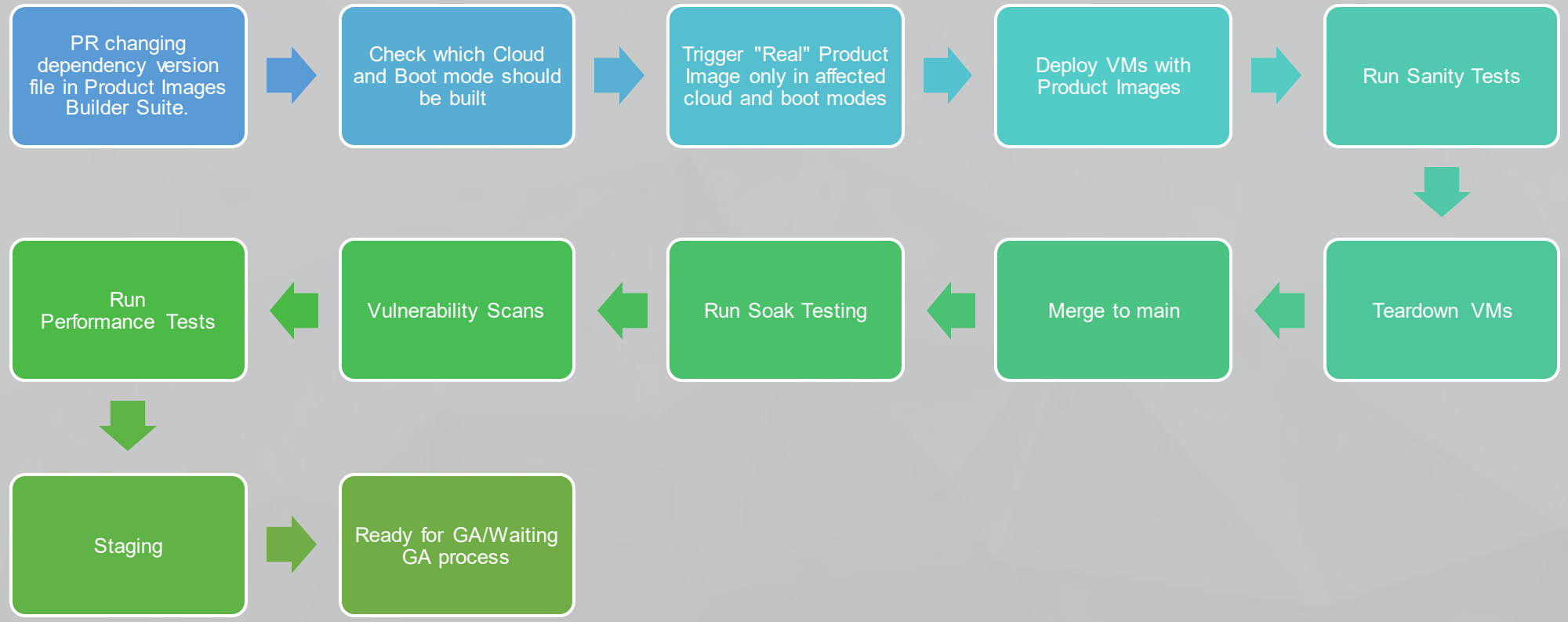
Workflow for Regression Tests: Examples

Regression Tests: Use Case



Workflow for Promotion Image: Example

Promotion Image: Use Case



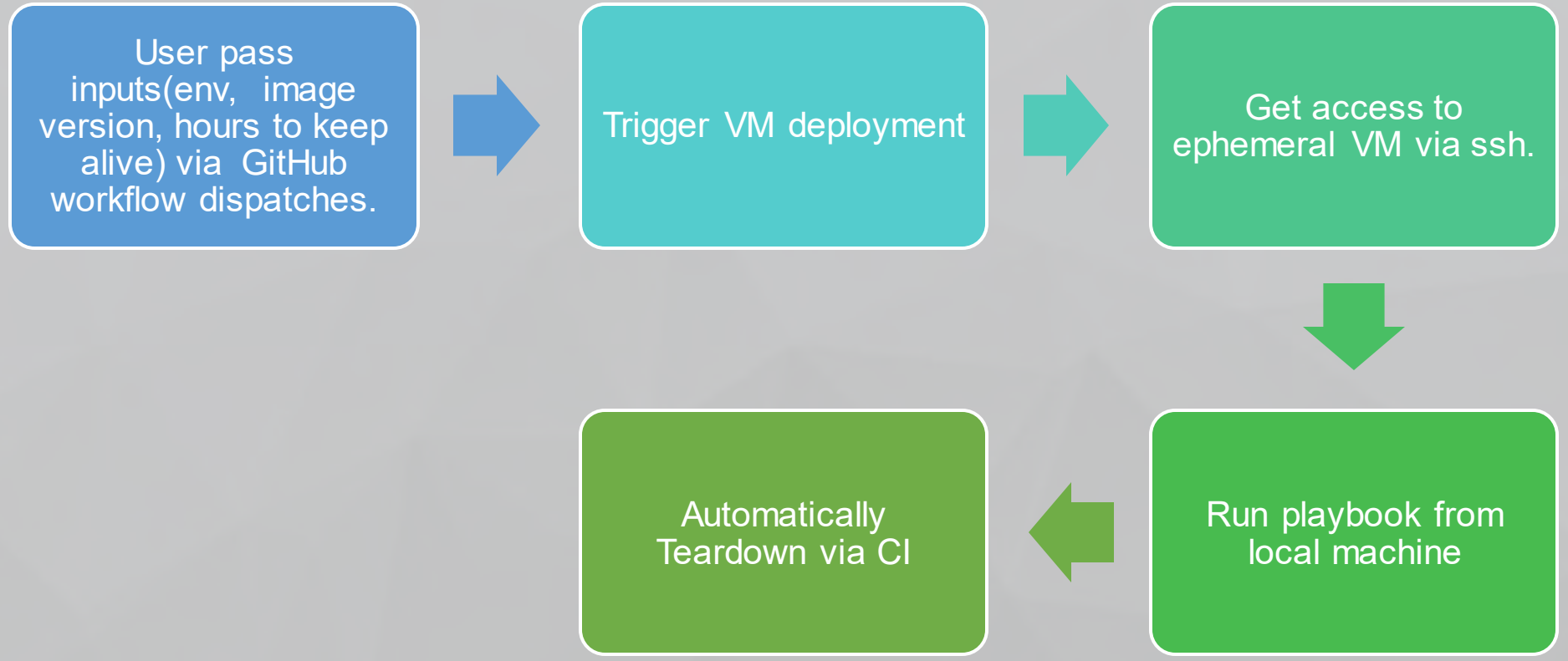
Dev Builds, VM debug and Manual Tests: Examples

Dev builds:
Use Case

User pass inputs(Seed/Base Image and other custom versions) via Github workflow dispacthes

Build a dev image to be used by the developer in manual deployment or manual tests.

Manual Tests: Use Case



Code Sample

GitHub Action – Repos Structure Samples

```
├── .github
│   ├── actions
│   │   ├── aws-build
│   │   │   └── action.yml
│   │   ├── azure-build
│   │   │   └── action.yml
│   │   ├── esx-build
│   │   │   └── action.yml
│   │   └── historical-images
│   │       └── action.yml
│   └── workflows
│       ├── ib-build-prod-images.yml
│       ├── ib-regression-tests.yml
│       ├── rw-build-prod-pipeline.yml
├── ansible
│   └── image.yml
├── packer
│   ├── aws
│   │   ├── bios
│   │   │   └── image.pkr.hcl
│   │   └── uefi_sb
│   │       └── image.pkr.hcl
│   ├── azure
│   │   ├── bios
│   │   │   └── image.pkr.hcl
│   │   └── uefi_sb
│   │       └── image.pkr.hcl
│   └── esx
│       ├── bios
│       │   └── image.pkr.hcl
│       └── common
│           ├── cloud-init
│           │   ├── meta-data
│           │   └── user-data
│           ├── download-base-image.pkr.hcl
│           └── templates
│               └── rc-agent-template.ovf
├── scripts
│   └── images_spec.sh
└── versions.yml
```

```
├── .github
│   ├── actions
│   │   ├── aws-test
│   │   │   ├── ansible-tests
│   │   │   │   └── action.yml
│   │   │   ├── setup-ac
│   │   │   │   └── action.yml
│   │   │   └── teardown-ac
│   │   │       └── action.yml
│   │   ├── azure-test
│   │   │   ├── ansible-tests
│   │   │   │   └── action.yml
│   │   │   ├── setup-ac
│   │   │   │   └── action.yml
│   │   │   └── teardown-ac
│   │   │       └── action.yml
│   │   └── esx-test
│   │       ├── ansible-tests
│   │       │   └── action.yml
│   │       ├── setup-ac
│   │       │   └── action.yml
│   │       └── teardown-ac
│   │           └── action.yml
│   └── workflows
│       ├── it-debug-ac.yml
│       ├── it-manual-tests.yml
│       ├── ib-regression-tests.yml
│       └── rw-test-pipeline.yml
└── ansible
    ├── callback_plugins
    │   └── check_failures.py
    └── tests.yml
```

Composite Action – AWS Build Sample

```yaml
name: aws-build
description: Build Base Image for AWS with Packer.

inputs:
  image-version:
    description: |
      The version to create a new base image.
      Example: '2412301350'
    required: true
  seed-image-name:
    description: |
      The name of the seed VM image used as the foundation for the Base Image.
      Example: 'CiscoHardened-Ubuntu22.04LTS-amd64-hvm-2024-04-01'
    required: true
  ref:
    description: |
      The branch, tag, or SHA to checkout.
      Default: 'main'
    default: main

outputs:
  image-name:
    description: |
      The name of the image version created by the build process.
    value: ${{ steps.image-version.outputs.image-name }}
```

```yaml
runs:
  using: composite
  steps:

    <other steps>

    - name: Build and push versioned image
      env:
        AWS_REGION: ${{ steps.init.outputs.region }}
        BRANCH: ${{ steps.branch.outputs.branch }}
        JOB_ID: ${{ github.run_id }}
        PACKER_GITHUB_API_TOKEN: ${{ env.GH_TOKEN_GITHUB_COM }}
        PACKER_SUBNET: ${{ env.SUBNET_ID }}
        PACKER_VPC: ${{ env.VPC_ID }}
        RELEASE_NAME: ${{ steps.image-version.outputs.image-name }}
        SEED_IMAGE_NAME: ${{ inputs.seed-image-name }}
      shell: bash
      run: |
        set -x
        packer init         packer/ami/ami.pkr.hcl
        packer validate     packer/ami/ami.pkr.hcl
        packer build -debug packer/ami/ami.pkr.hcl
```

Reuseable Workflow – Build Sample

```yaml
name: '[RW][Prod] Build Pipeline'

on:
  workflow_call:
    inputs:
      image-version:
        description: |
          The version to create a new base image.
          Example: '2412301350'
        type: string
        required: true
      seed-image-name:
        description: |
          The name of seed VM image used as the foundation for the Base Image.
          Example: 'seed-image-2412301355'
        type: string
        required: true
      platform:
        description: |
          The platform for which the VM image is built.
          Example: 'aws', 'azure', 'gcp', 'qcow2'
        type: string
        required: true
      ref:
        description: |
          The branch, tag, or SHA to checkout.
        type: string
        default: main

    outputs:
      image-name:
        description: |
          The image name created by the build.
          Example: 'base-image-2412301350'
        value: ${{ jobs.build-image.outputs.image-name }}
```

```yaml
jobs:
  build-image:
    name: Build
    runs-on:
      - self-hosted
      - x64
      - env:ops-prod
    timeout-minutes: 120

    outputs:
      image-name: ${{ steps.export.outputs.image-name }}

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Create action-with string
        id: create-action-with
        run: |
          action_with=$(echo '{
            "image-version": "${{ inputs.image-version }}",
            "seed-image-name": "${{ inputs.seed-image-name }}",
            "ref": "${{ inputs.ref }}"
          }' | jq -c)

          echo "action-with=${action_with}" >> $GITHUB_OUTPUT

      - name: Run Build
        id: run-build
        uses: my-org/reusable-workflows/.github/actions/dynamic-action@main
        with:
          action-uses: my-org/-base-image-builder-suite/.github/actions/${{ inputs.platform }}-build@${{ inputs.ref }}
          action-with: ${{ steps.create-action-with.outputs.action-with }}
```

Calling Reuseable Workflow – Build Sample

```yaml
- name: Create PR images spec
  shell: bash
  run: |
    source scripts/images_spec.sh
    create_images_spec images_spec.yml
    update_images_spec versions.yml images_spec.yml
    create_encoded_images_spec images_spec.yml images_spec_encoded.txt
    cd main-branch
    create_images_spec images_spec.yml
    update_images_spec versions.yml images_spec.yml
    create_encoded_images_spec images_spec.yml images_spec_encoded.txt
    cd ../
    create_encoded_build_spec main-branch/images_spec_encoded.txt images_spec_encoded.txt images_spec_to_build.txt

- name: Set up matrix
  id: set-up-matrix
  shell: bash
  env:
    TIMESTAMP: ${{ steps.timestamp.outputs.timestamp }}
  run: |
    dicts=()
    while IFS= read -r line; do
      decoded_json=$(echo "$line" | base64 -d)

      platform=$(echo "$decoded_json" | jq -e -r '.platform')
      boot_mode=$(echo "$decoded_json" | jq -e -r '.boot_mode')
      key="${platform}_${boot_mode}"
      image_version="${TIMESTAMP}"

      merged_json=$(echo "$decoded_json" | jq --arg image_version "$image_version" '. + {"image_version": $image_version}')

      dicts+=($(echo "{\"$key\": $merged_json}" | jq -c))
    done < images_spec_to_build.txt

    jobs=$(echo "${dicts[@]}" | jq -s add | jq -c)
    echo "jobs=${jobs}" >> $GITHUB_OUTPUT
```

Calling Reuseable Workflow – Build Sample

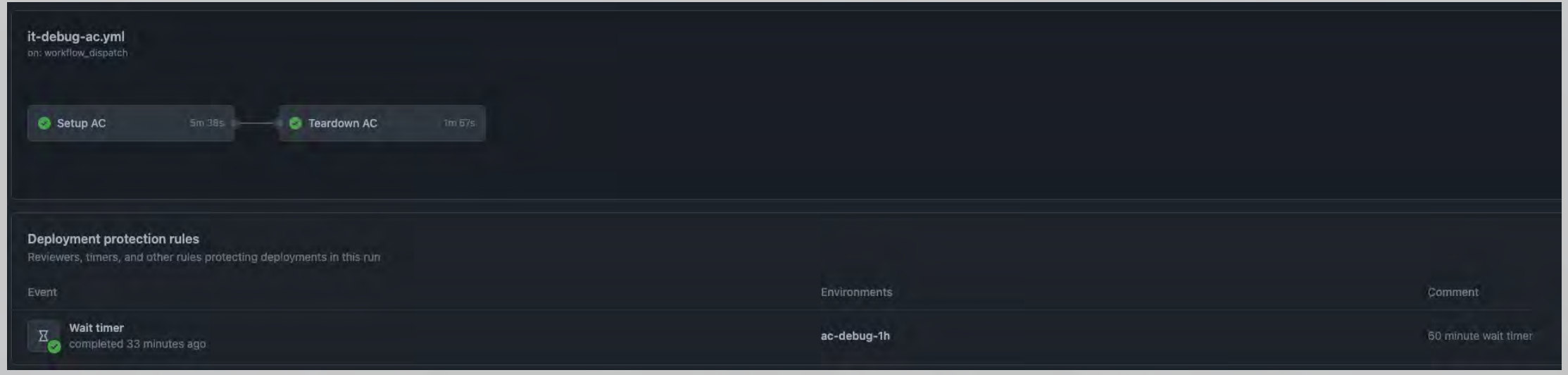
```yaml
pipeline-aws-bios:
  name: AWS bios
  needs: generate-jobs
  if: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_bios'] }}
  uses: ./.github/workflows/rw-build-prod-pipeline.yml
  with:
    platform: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_bios']['platform'] }}
    boot-mode: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_bios']['boot_mode'] }}
    image-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_bios']['image_version'] }}
    base-image-name: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_bios']['base_image_name'] }}
    manifest-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_bios']['manifest_version'] }}
    upgrade-helper-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_bios']['upgrade_helper_version'] }}
    ref: ${{ github.head_ref }}

pipeline-aws-uefi_sb:
  name: AWS uefi_sb
  needs: generate-jobs
  if: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_uefi_sb'] }}
  uses: ./.github/workflows/rw-build-prod-pipeline.yml
  with:
    platform: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_uefi_sb']['platform'] }}
    boot-mode: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_uefi_sb']['boot_mode'] }}
    image-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_uefi_sb']['image_version'] }}
    base-image-name: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_uefi_sb']['base_image_name'] }}
    manifest-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_uefi_sb']['manifest_version'] }}
    upgrade-helper-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['aws_uefi_sb']['upgrade_helper_version'] }}
    ref: ${{ github.head_ref }}

pipeline-azure-bios:
  name: Azure bios
  needs: generate-jobs
  if: ${{ fromJson(needs.generate-jobs.outputs.jobs)['azure_bios'] }}
  uses: ./.github/workflows/rw-build-prod-pipeline.yml
  with:
    platform: ${{ fromJson(needs.generate-jobs.outputs.jobs)['azure_bios']['platform'] }}
    boot-mode: ${{ fromJson(needs.generate-jobs.outputs.jobs)['azure_bios']['boot_mode'] }}
    image-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['azure_bios']['image_version'] }}
    base-image-name: ${{ fromJson(needs.generate-jobs.outputs.jobs)['azure_bios']['base_image_name'] }}
    manifest-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['azure_bios']['manifest_version'] }}
    upgrade-helper-version: ${{ fromJson(needs.generate-jobs.outputs.jobs)['azure_bios']['upgrade_helper_version'] }}
    ref: ${{ github.head_ref }}
```

Regression Tests in Base Image – Pipeline Sample

Debug VM – Pipeline Sample



Crucial Takeaways

- Developers don't need to deploy the ephemeral infra themselves.
- In the worst-case scenario, all stages and images are built and fully tested within 3 hours.
- We can easily add more cloud providers, and it will not affect the build/testing time.
- Using IaC principles alongside pinned versions allows us to find bugs faster.
- The pipeline is modular, so differences between cloud providers are self-contained.

More about the stack

- https://docs.github.com/en/actions/using-workflows/reusing-workflows
- https://docs.github.com/en/actions/creating-actions/creating-a-composite-action
- https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows
- https://developer.hashicorp.com/packer/tutorials/hcp-get-started
- https://developer.hashicorp.com/terraform/tutorials?product_intent=terraform
- https://docs.ansible.com/ansible/latest/getting_started/index.html

Where to find me?

@edersonbrilhante

@edersonbrilhante

Thank You