



All Clouds

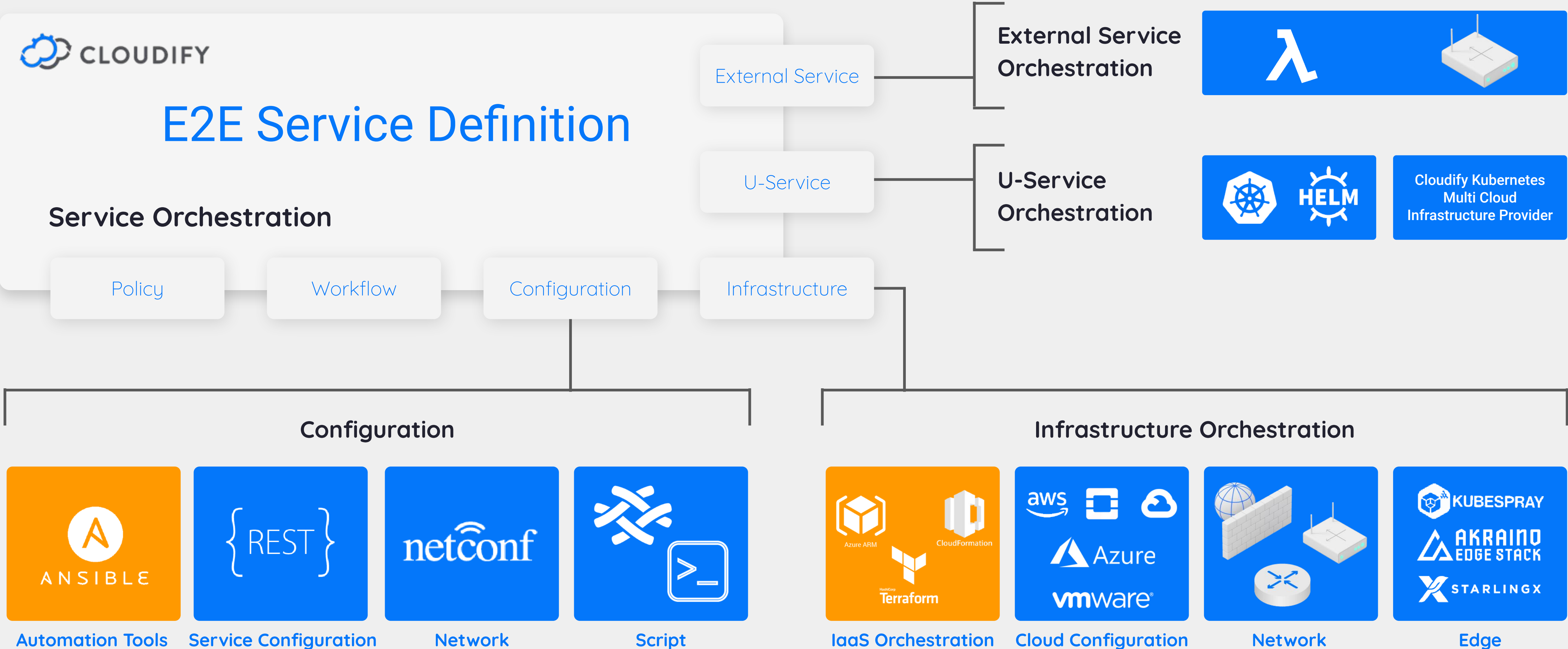
Any Service

Unlimited Locations

Cloudify and Terraform Integration



Cloudify Infrastructure Orchestration Plugins



Cloudify & Terraform Integration

Managing Terraform as a first class citizen within Cloudify Manager infrastructure

Integration & installation

Install Terraform binary and modules

Parameters & secret management

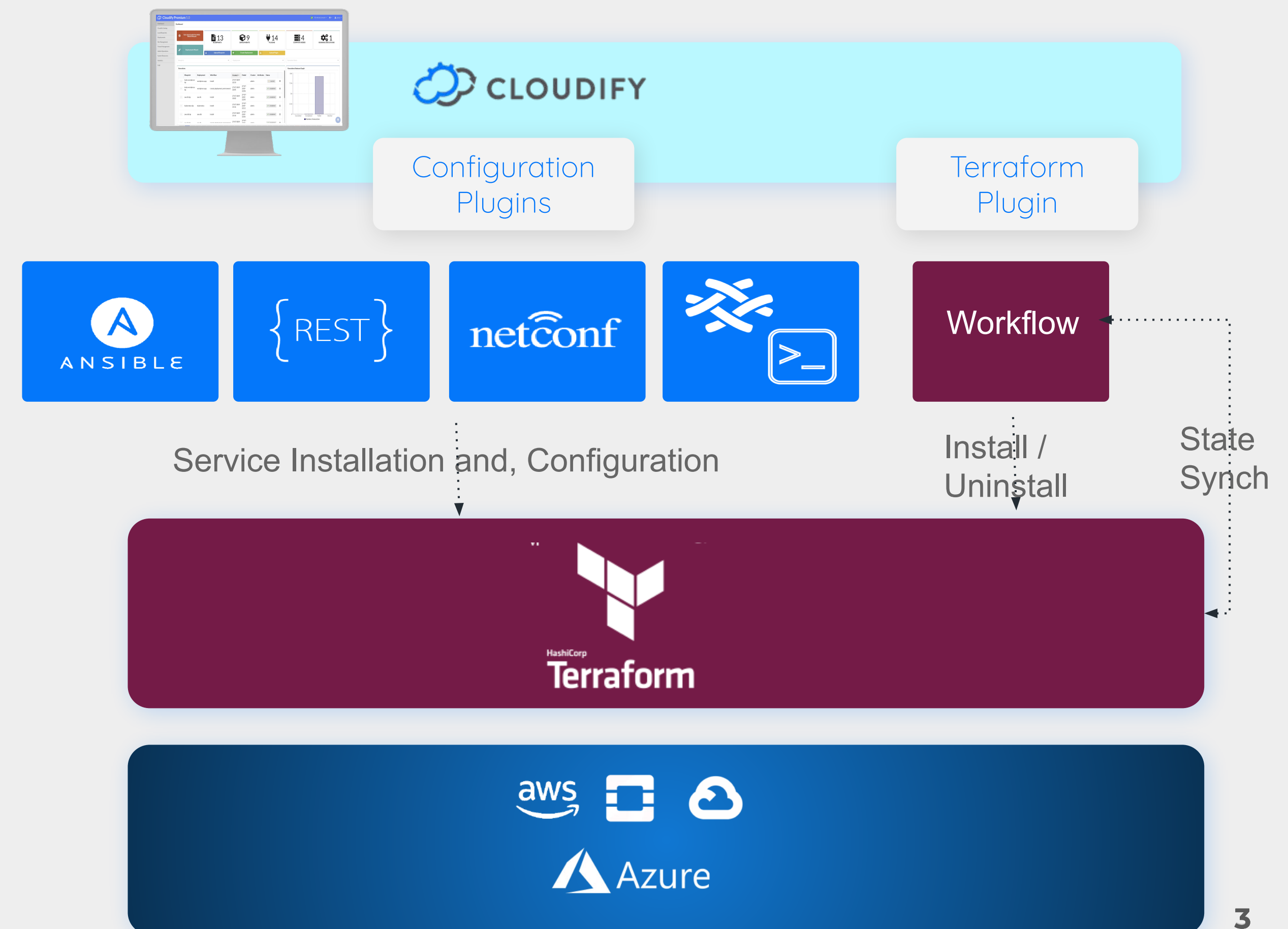
Pass inputs to modules through cloudify secrets and input

Log aggregation

Index all the logs under a common log trail

Passing Terraform state properties to other services through Cloudify Capabilities :

Save and share Terraform state properties (IP, tags,..) **implicitly** with other non-Terraform services through service relationship



Cloudify & Terraform Integration

Day 2 Operation and Workflow Management.
Allow Concurrent execution (avoid race condition)

Refresh Terraform State (State Pull)

Install Terraform binary and modules

Reload Terraform Template

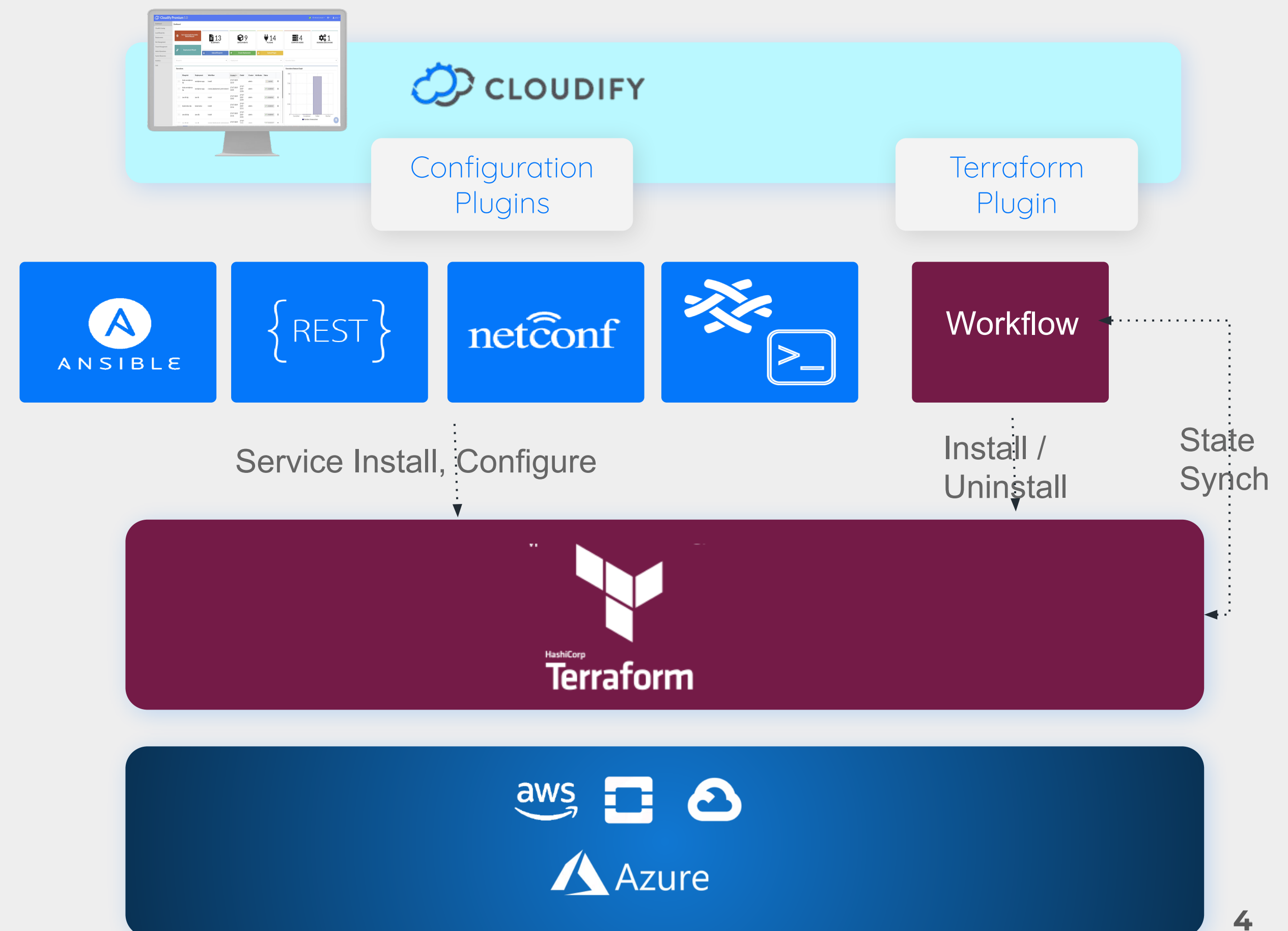
Update terraform state from updated templates

Configuration Management

Configure the infrastructure using Fabric, Ansible , Agent based method

Full Life Cycle Management

Manage Terraform installation and uninstallation as part of the end-to-end service automation



Cloudify & Terraform Integration

Management UI

Topology view

View Terraform infrastructure as part of the overall environment topology *

Workflow monitoring

View the execution graph of Terraform and non Terraform tasks as well as between multiple Terraform infrastructure services.

Composer

Simplify the composition of terraform template artifacts using a drag n drop experience *

State monitoring - Continuously monitor the state of Terraform infrastructure *

The screenshot displays the Cloudify Spire 5.0 Management UI. On the left, a blueprint editor shows a YAML file named 'blueprint.yaml' with the following content:

```
tosca_definitions_version: cloudify_dsl_1_3
description: >
  This blueprint creates infrastructure on AWS using Terraform.
inputs:
  - http://cloudify.co/spec/cloudify/5.0.5/types.yaml
  - plugin:cloudify.terraform.plugin
inputs:
  terraform_installation_source:
    description: >
      Where to get Terraform from.
    type: string
    default: "https://releases.hashicorp.com/terraform/0.12.21/terraform-0.12.21-linux-amd64.zip"
  terraform_plugins:
    type: list
    default:
      - "https://releases.hashicorp.com/terraform-provider-template/2.0.0/terraform-provider-template_2.0.0_linux_amd64.zip"
      - "https://releases.hashicorp.com/terraform-provider-aws/2.49.0/terraform-provider-aws_2.49.0_linux_amd64.zip"
  terraform_executable:
    type: string
    default: "/tmp/terraform/bin/terraform"
  terraform_plugins_dir:
    type: string
    default: "/tmp/terraform/plugins"
  terraform_storage_path:
    type: string
    default: "/tmp/terraform/storage"
  module_source:
    type: string
  admin_user:
    description: >
      The admin user name for the created VM.
    type: string
    default: "centos"
node_templates:
  terraform:
    type: cloudify.nodes.terraform
    properties:
      use_existing_resource: false
      installation_source: { get_input: terraform_installation_source }
      plugins: { get_input: terraform_plugins }
      executable_path: { get_input: terraform_executable }
      plugins_dir: { get_input: terraform_plugins_dir }
      storage_path: { get_input: terraform_storage_path }
  cloud_resources:
    type: cloudify.nodes.terraform.Module
    properties:
```

On the right, the 'Deployment Topology' view shows a graph with two main components: 'cloud_resources' and 'terraform'. Below this, the 'Deployment Executions' section shows a workflow graph for a deployment created on 07-10-2019 11:43. The workflow includes tasks such as 'Install port ny6ehp', 'Neutron plugin port create', 'Relationship lifecycle establish subgraph', 'Install cloudify host cloud config vgn6t', 'Cloudify cloudinit tasks update', and 'Install host cs23ft'. The status of each task is indicated by green boxes for 'succeeded' and grey boxes for 'pending'.

*work in progress

Cloudify & Terraform Integration

Decouple the application from the infrastructure orchestration choice and allow interoperability

Avoid Costly Blueprint Transformation

Build upon existing automation artifacts where possible.
Use the right orchestration tool for the job

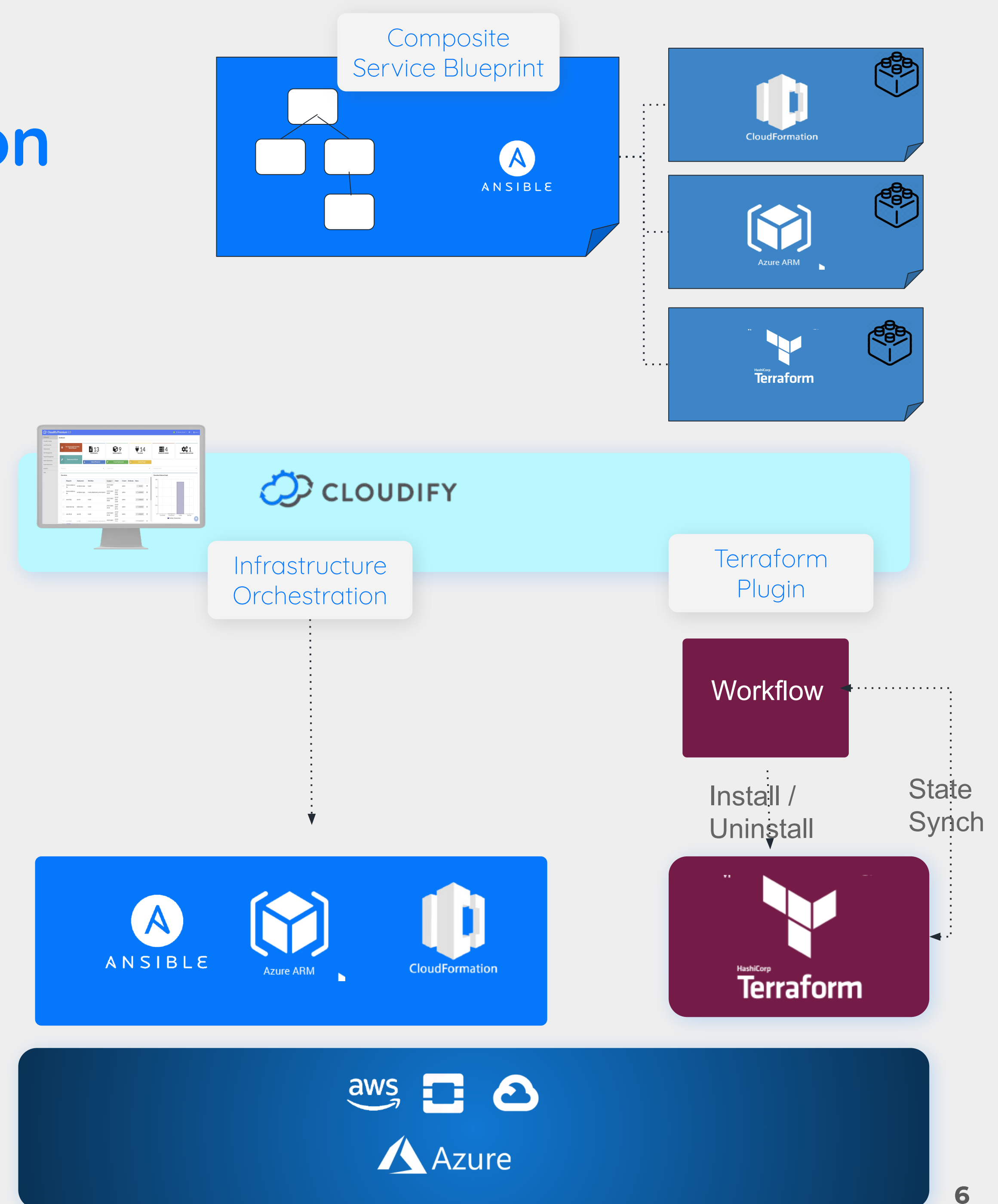
Interoperability between Orchestration Platforms

Define relationship and pass content information between different Terraform, Cloud Formation, Azure ARM, TOSCA, Ansible templates.

End to End Automation

All the automation components under a single automation experience. Faster time to market.

Support AWS Cloud Formation, Azure ARM, Ansible ..



Terraform Demo

```
1 tosca_definitions_version: cloudify_dsl_1_3
2
3 description: >
4   This blueprint creates infrastructure on AWS using Terraform.
5
6 imports:
7   - http://cloudify.co/spot/cloudify/5.0.5/types.yaml
8   - plugin:cloudify.terraform.plugin
9
10 inputs:
11   terraform_installation_source:
12     description: >
13       Where to get Terraform from.
14     type: string
15     default: 'https://releases.hashicorp.com/terraform/0.12.21/terraform-0.12.21-linux-amd64.zip'
16   terraform_plugins:
17     type: list
18     default:
19       - 'https://releases.hashicorp.com/terraform-provider-template/2.0.0/terraform-provider-template_2.0.0_linux_amd64.zip'
20       - 'https://releases.hashicorp.com/terraform-provider-aws/2.49.0/terraform-provider-aws_2.49.0_linux_amd64.zip'
21   terraform_executable:
22     type: string
23     default: '/tmp/terraform/bin/terraform'
24   terraform_plugins_dir:
25     type: string
26     default: '/tmp/terraform/plugins'
27   terraform_storage_path:
28     type: string
29     default: '/tmp/terraform/storage'
30   module_source:
31     type: string
32   admin_user:
33     description: >
34       The admin user name for the created VM.
35     type: string
36     default: 'centos'
37
38 node_templates:
39   terraform:
40     type: cloudify.nodes.terraform
41     properties:
42       use_existing_resource: false
43       installation_source: { get_input: terraform_installation_source }
44       plugins: { get_input: terraform_plugins }
45       executable_path: { get_input: terraform_executable }
46       plugins_dir: { get_input: terraform_plugins_dir }
47       storage_path: { get_input: terraform_storage_path }
48
49   cloud_resources:
50     type: cloudify.nodes.terraform.Module
51     properties:
```

Cloudify Spire 5.0

Local Blueprints > If example > example.dip

Execute workflow Update deployment Delete deployment

Deployment Topology

cloud_resources terraform

Deployment Nodes

Name	Type	Contained in	Connected to	Host	Creator	# Instances	Groups
terraform	cloudify.nodes.terraform				admin	1	
cloud_resources	cloudify.nodes.terraform.Module				admin	1	

Terraform Demo

Cloudify & Terraform Integration

Leveraging Cloudify DB clustering and geo-redundancy to manage TF state

Terraform Installation and Modules Clustering *

Terraform is installed on the manager file system. The installation and modules are replicated across all the managers in the cluster.

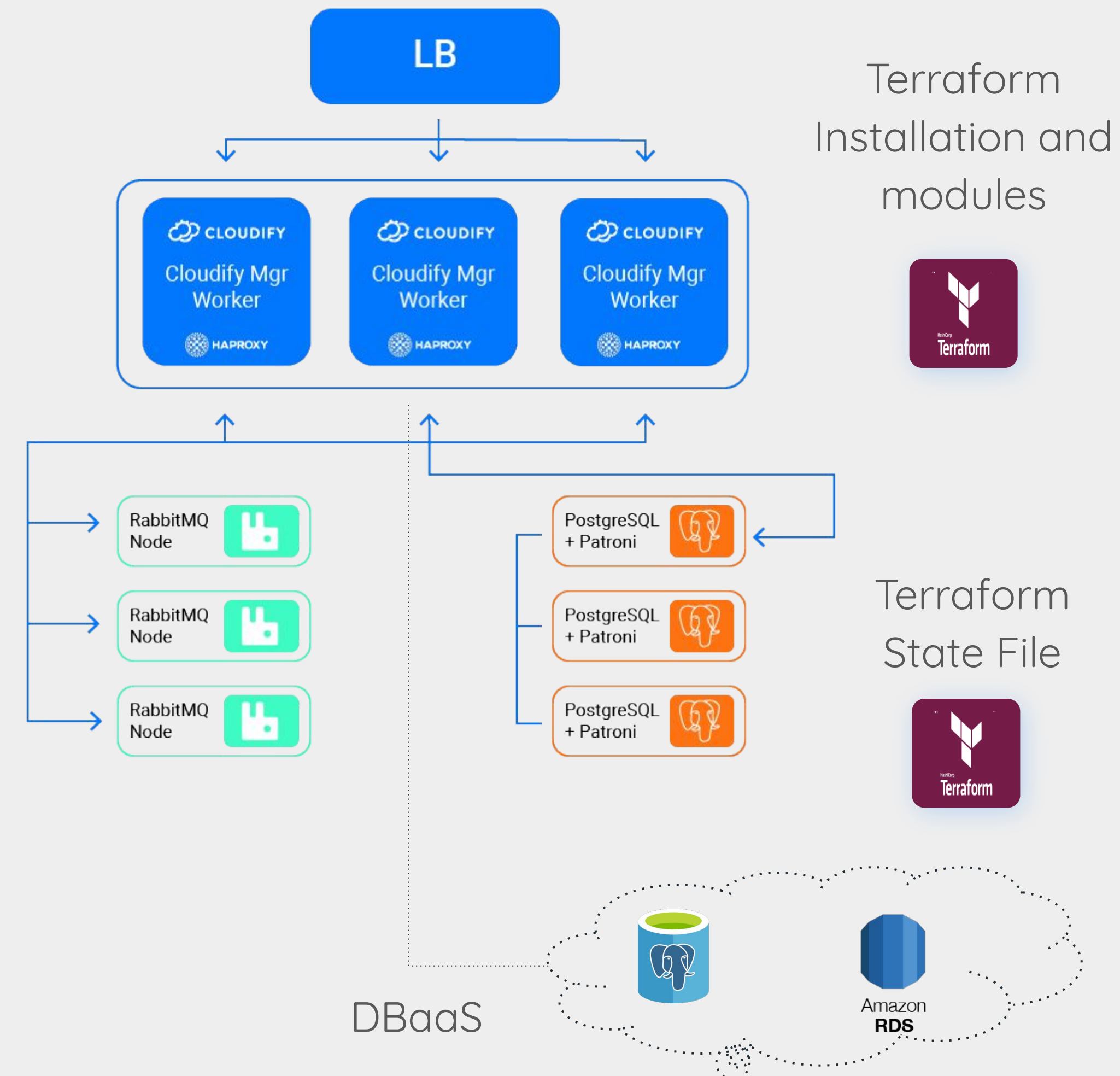
Terraform state file replication

Terraform state file is stored in the Cloudify DB via runtime properties. Before each execution the state file is fetched to the local directory and uploaded again upon completion.

Leverage Local DB Cluster, Geo Redundancy and DBaaS

Leverage Local, Multi-Site and Cloud based DB clustering configuration to allow flexible TF state clustering.

*requires customisation of filesystem replication configuration



Future Direction and vision

Manage Service Composition between Multiple Distributed Terraform Templates

Manage the relationship between Terraform Templates

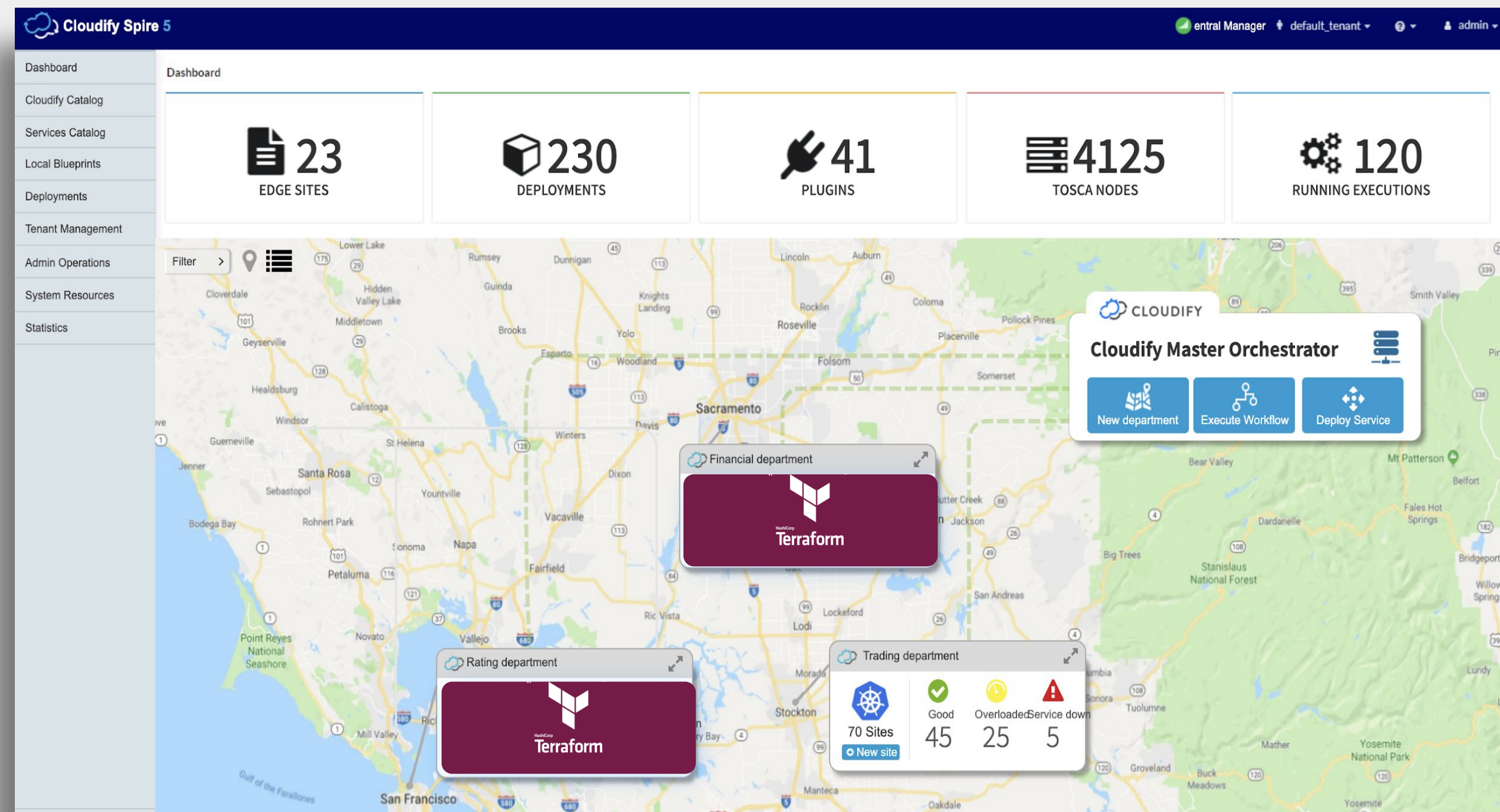
Define relationship and pass context information between different Terraform templates.

Multi Site / Department Management

Manage multiple Terraform installation and management across multiple department and sites from a single point of access.

UI Improvements

New topology, workflow and composer



The Big Picture..

Introducing Cloudify EaaS to simplify CI/CD pipeline in a multi cloud environments

New Service Composition

Allowing dynamic binding between services, adding/removing new services on the fly. Cascading workflow, Shared resources..

Kubernetes Support

- Plugin: Improved state reporting, refresh/update.
- Platforms: OpenShift, KubeSpray, GKE, EKS, AKS.

Ansible Plugin Update

Integration with remote repository management, e.g. Git. Remote state refresh and update.

Jenkins Plugin

Creating multi-cloud environment using Cloudify as a native build step. Decoupling the application from the infrastructure.





Thank
You.