OVERVIEW

APCERA ARCHITECTURE

Apcera Foundational Whitepaper • October 2015

APCERA

Table of Contents

CHAPTER 1
INTRODUCTION

Apcera's platform is the world's first trusted, policy-driven cloud platform supporting deployment, orchestration and governance for diverse workloads across multiple clouds (hybrid, public and private). Policy is pervasive and foundational. Delivering more than just role-based access control, policy specifies and governs the behavior of the platform using declarative rules. Policies span across each layer in the system—from physical access to the networking layer to the discovery and awareness of communication protocols.

This enables enterprises to accelerate cloud adoption, taking advantage of the speed, performance, scalability and cost-effectiveness of combining cloud environments, while ensuring IT operational governance and compliance for mission-critical applications and data. It also provides the most secure deployment of Docker containers.

The Apcera platform allows DevOps and ITOps to apply and enforce policy across the entire workload lifecycle, ensuring what should happen will happen, and what is not permitted to happen cannot happen.

Apcera delivers all necessary elements of cloud application management—including deployment, security, governance, policy, workload management, auditing and reporting—delivered seamlessly across single cloud, multi-cloud and hybrid cloud environments. The Apcera trusted cloud platform balances workloads, automatically remaps moved applications, and restarts and reconnects failed applications, ensuring your applications run smoothly even in the presence of hardware and software failures.

FIGURE 1 | Diverse workloads supported by Apcera's platform.

Chapter 1: Introduction (cont'd)

Apcera can deploy a diverse set of workloads across your target infrastructures. It can do so without requiring changes to the applications or workloads, for example:

- Application code written in Java, Ruby, PHP, Go, Node.js, Perl, Python, Ruby/Rails, Bash, and other languages. You can run custom workloads by writing a workload stager.
- Executable application packages built by CI systems and standardized deployable archives such as Web Application Resource (WAR) bundles.
- Docker images from any source such as the Docker Hub or your corporate repository.
- Bare OS to enable building custom application environments.

The Apcera platform dynamically assembles user-created workloads, services running within the platform, and services running outside the platform into coherent enterprise applications. This paper will explain how Apcera discovers, addresses, connects and load balances between internal and external workloads by using unique strategies such as binding, credential remapping and semantic awareness.

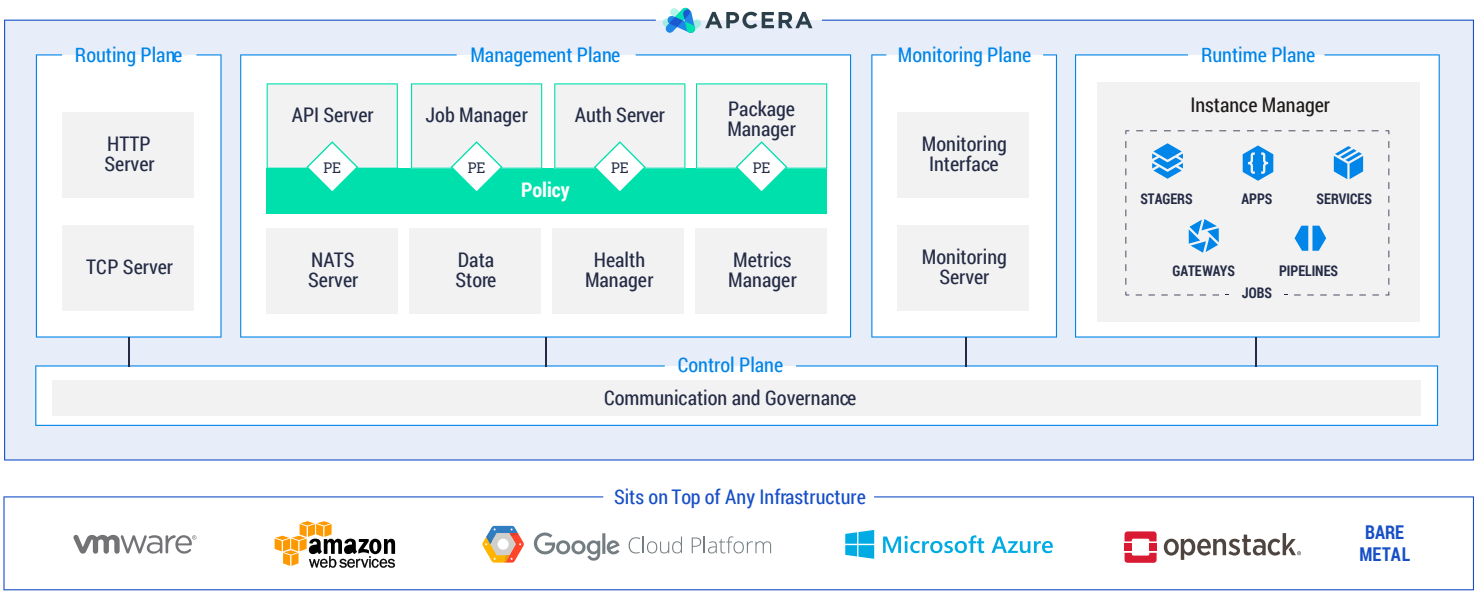
This paper also explains how the Apcera Policy Engine governs your workloads using tactics such as:

- Attribute based access control (ABAC) of all resources
- Dependency rules for binary packages
- Quota and configuration settings
- Database access protection
- Semantic rules for app-to-service communication

CHAPTER 2
APCERA INFRASTRUCTURE COMPONENTS

ARCHITECTURE

FIGURE 2 | Apcera Architecture



The router sends commands from APC and Console to the Auth and API server. It routes HTTP/S and TCP traffic between external clients and internal running applications. All Apcera components interact with each other securely using the NATS high-performance messaging system. The Policy Engine (PE) enforces your policies.

CONSOLE

The Apcera console is a user interface accessible from a web browser. Each cluster provides a unique web console instance for interacting with the jobs, resources, and policies in that cluster. The console uses XHR to asynchronously load JSON data from the API Server (AJAX). You can use the web console to visualize, browse and manage the Apcera platform and you can enable and disable live updating.

APC

A command line interface (CLI) tool for interacting with Apcera's platform.

NATS

All Apcera components interact with each other securely using the NATS high-performance messaging system. NATS is a lightweight

publish/subscribe and distributed queuing (interest/subscribe) system. It does not have persistence or transactions. NATS has built-in primitives to actively prune the interest graph during the receipt of answers from an unknown group size, which offloads the clients from large unpredictable CPU spikes when throwing away messages.

ROUTER

A route is the manner in which you expose a workload or application in Apcera's platform by giving it an externally-reachable hostname like www.example.com. A defined route can be consumed by the router to provide connectivity from external clients to your applications. Many components work together to create the Apcera platform. The router provides external hostname mapping and load balancing. The Apcera router is based on Nginx and supports the following protocols:

- HTTP
- HTTPS
- WebSockets
- SSL/TLS

Apcera automatically updates the router configuration when any relevant changes happen in the system. The router also forwards commands from APC and Console to the API server. A separate router handles TCP based communications.

API SERVER

Commands from Apcera control programs, APC and Console, go to the API server, which forwards them to the appropriate component as remote procedure call (RPC) messages on NATS. These commands can initiate staging, start jobs, bind jobs to services, query the system for information about jobs and resources, and perform any other management tasks.

For example, if a user changes the number of instances for a Job, via APC or Console, the API Server identifies that the property has changed and issues the proper RPC calls to the Job Manager to add new instances or remove instances.

JOB MANAGER

The Job Manager is the authoritative source on what the state and configuration of the jobs should be and informs Instance Managers and the Health Managers of changes in the expected state of the system, including numbers of instances, routes and packages. Health Manager and Instance Manager are responsible for reflecting the state in the runtime environment.

PACKAGE MANAGER

Packages are the output of—and some of the inputs to—staging. The Package Manager preserves packages (application code, operating systems, runtimes and dependencies) and delivers them on demand to system components like Stagers and Instance Managers.

AUTHENTICATION SERVER

The Auth Server stores and distributes policies and issues auth tokens. Tokens are like passports that control movement and access of everything inside the system.

The Auth Server reads the policies to determine which requesters should receive tokens and what those tokens should allow. Each component evaluates policies at runtime to determine whether or not to allow a requested operation.

POLICY ENGINE

The Policy Engine is distributed across all Apcera components—API Server, Package Manager, Job Manager, Instance Managers, Health Manager, Auth Server and Metrics Manager. Each component has access to all policy information and performs policy-related tasks. Apcera uses a system of resource identifiers known as fully qualified names (FQNs), each consisting of a resource type, a namespace and a location within the namespace. Apcera policies associate permissions with a realm. The form of the permissions depends on the resource type. The permissions usually allow an operation specific to that resource type for a specified user or class of users. Any operation that is not specifically allowed is disallowed.

Examples of constraints that you can specify by policy include, but are not limited to:

- Node.js applications running in production must use Node.js 0.10.25.
- Applications running in production can only use databases or other services running in production.
- Applications running in the testing environment cannot connect to services running in production.
- Applications that access the web must be scanned for vulnerabilities issues and malware.
- Developers can use the runtime of their choice for applications and services running in their own sandbox but not when running in production.
- Set the maximum RAM quota for each developer.
- Disallow specific software version from being deployed.
- Identify if any existing applications are out of compliance.

FIGURE 3 | Policy Engine

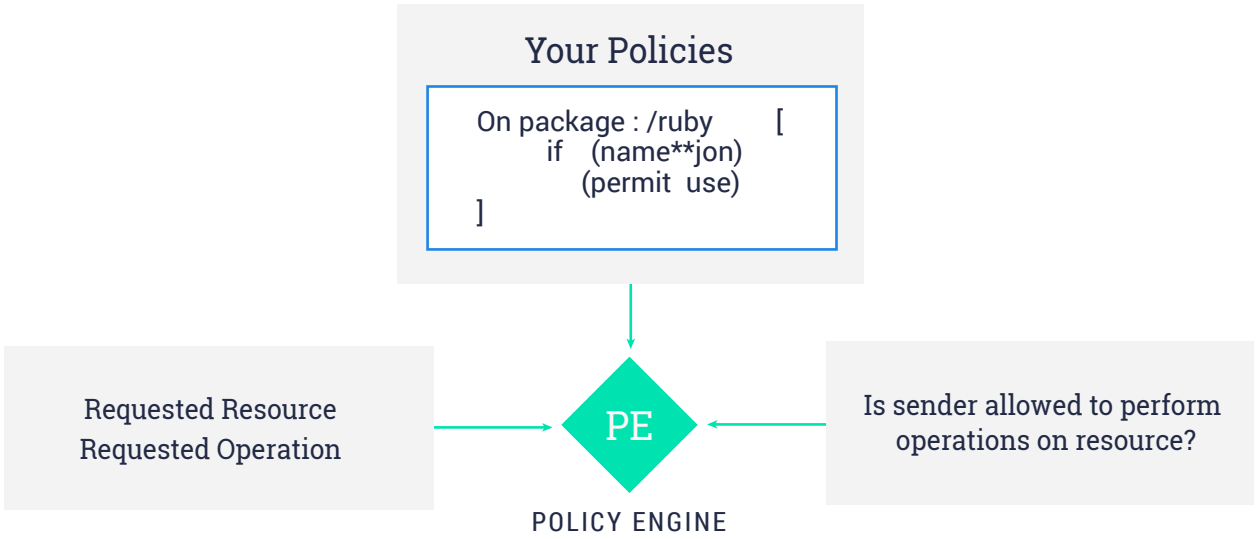
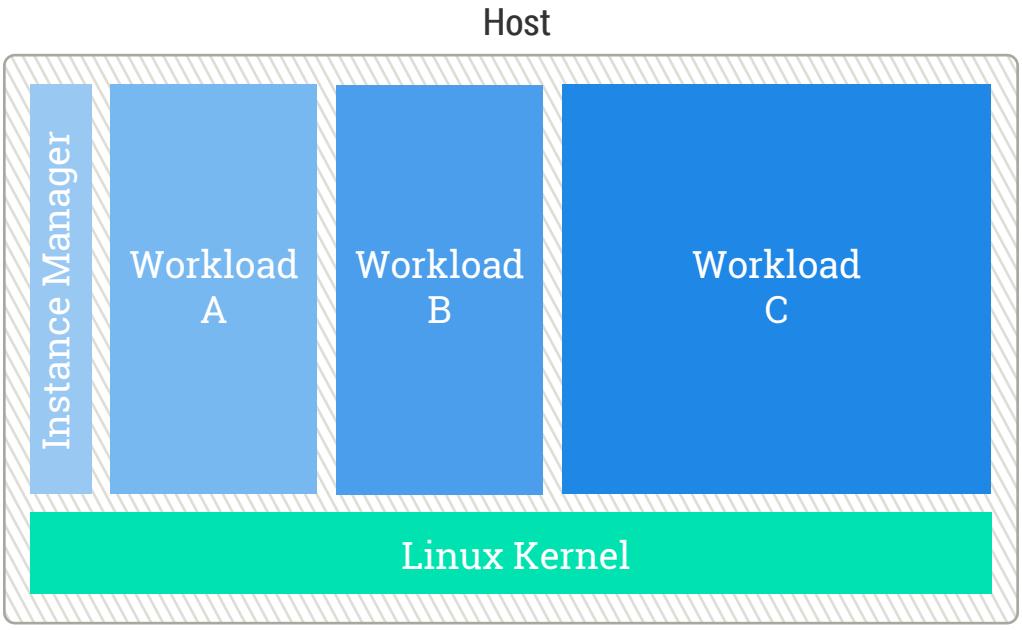


FIGURE 4 | Instance Manager



INSTANCE MANAGER

The Instance Manager is installed on the servers (physical or virtual systems) dedicated to host workloads. The Instance Manager is responsible for running and monitoring the workloads. Every workload (application, operating system, Docker container, stager, service gateway) in Apcera is defined as a job. For jobs requiring a semantic pipeline, the Instance Manager is also responsible for instantiating the appropriate semantic pipeline.

HEALTH MANAGER

The Health Manager monitors the state of the system and compares it with the expected state, ensuring that the configuration in the Job Manager matches what is actually run by the Instance Managers. It ensures the proper number of job instances are running, with the current configuration, and ensures any stalled jobs are cleaned up. For example, if it expects ten instances of a certain job and only eight are running or healthy, it reports the discrepancy to the Job Manager so that it can correct those discrepancies.

METRICS MANAGER

The Metrics Manager gathers data about a job's CPU, memory, disk, and network usage rates. The Metric Manager also provides an API to enable users to make queries about these measures at any time.

ORCHESTRATOR

You use the Apcera Orchestrator tool to install, deploy, scale and upgrade your Apcera cluster. The Orchestrator tool interacts directly with the infrastructure layer and can deploy the Apcera platform to AWS, OpenStack, vSphere and Google. Using the Orchestrator tool, you can specify which Apcera processes can run on what machines and how those machines should be created.

CHAPTER 3
APCERA CORE CONCEPTS

This section gives a high-level view of how the parts of the platform work together

JOBS

The central Apcera concept is a job, a workload that Apcera runs, manages, and monitors, enabling you to update, report on and govern. Uploaded code can be anything from small applications, Docker containers to complex operating systems. They are all jobs within the system. Apcera components like Stagers, Service Gateways, and Semantic Pipelines—as well as components you write—are also run as jobs. Everything that runs in Apcera is a job.

Apcera ensures that your code has an appropriate execution environment and then packages it in a container, called an isolation context, within that environment so it is ready to run and deploy. This standardized form enables Apcera to deploy and orchestrate jobs in a uniform, policy-controlled way. Fine-grained policies, set by you, control a job's communication with other entities—within or outside the Apcera platform.

POLICY

Administrators can manage every aspect of the Apcera platform in a flexible and simple manner by deploying a set of policies that govern its behavior. Administrators define and update the policies to be enforced via the management console or the APC command line. Resulting policies are stored in the system. When new policies are added or existing ones are changed, the Policy Engine interprets the policies and communicates them to the system. Every component is policy-aware and can execute (enforce) the different policies. Policy rules are themselves distributed to the individual components of the system in a secure and trusted manner by using authenticated and encrypted NATS messages.

Policies control the way applications collaborate with other applications and data repositories (e.g. database backends) in the system by using a two-way handshake. Policies specify who applications connect to and who is allowed to connect to backends . Policies manage network ingress/egress rules, resource quotas (CPU, Memory, Disk Space, Network Rate), which versions of packages or base images for containers can be used, as well as which are deprecated or which jobs can depend on which packages.

Policies manage linking permissions for services and workloads, placement of multi-cloud workloads, application scaling and overall permissions for users, actions and components. Policies govern how users are identified and authorized to use services. Policies manage user roles and permissions. Policies are context-aware to allow or prevent actions over a certain threshold, time, date, target operation and target resource, allowing you to build powerful, attribute-based access control. You can use policies to define environments (i.e. Development, Test, Staging and Production) and ensure that anything that runs inside the platform is trusted.

NAMESPACE

An Apcera namespace provides a mechanism to scope resources in the cluster. Namespaces can be set locally for each user and are used when setting policy permissions. Namespaces are integral to configuring the behavior of a cluster and the jobs running in that cluster.

REALM

Apcera uses a system of resource identifiers called fully qualified names (FQNs), each consisting of a resource type and a location within the namespace. Because the namespace and location are hierarchical—like file system names—the leftmost part of an FQN defines a set of resources called a realm.

For example, auth::/oauth2/http defines the realm of access tokens for communicating with the Apcera API. The scope of a realm can be:

- All resources of a type in the cluster (job::/)
- All resources of a type in a namespace (job::/name/space)
- An individual resource (job::/name/space::job/name)

Apcera policies associate permissions with a realm. The form of the permissions depends on the resource type. The permissions allow an operation specific to that resource type for a specified user or class of users. Any operation that is not specifically allowed is disallowed.

PACKAGE

A Package is a collection of binary data and metadata that make up an application or service. For example, the Ubuntu operating system, the Java runtime, a Ruby on Rails application or a Bash script.

- Binary data: Apcera determines at runtime if the data is gzip'd, bzip'd or a flat tar file.
- Metadata: A JSON file that includes dependencies and provides environment variables along with other information.

STAGER

A Stager is a job that is started when a new package, application code or Docker image is uploaded into the system. Stagers examine the code and config files to determine dependencies, consider any restrictions you have placed on use of OSs and runtimes, and select the most appropriate OS and runtime from those available within Apcera's Package Manager. Stagers combine these with your code into a ready-to-run package and give it to the Package Manager. The Package Manager saves it and provides it whenever a new instance of your app must be created.

Apcera provides you with the Stager API for writing custom stagers. For example, you can write a stager to run RSpec on any Ruby application deployed to production. RSpec is a testing tool for the Ruby programming language. Stagers can be written in the language of your choice and deployed like any other application. Once deployed, Stagers can be used to stage other packages. Examples of packages include the Ubuntu operating system, the Java runtime, a Ruby on Rails application or a Bash script.

FIGURE 5 | Package Manager

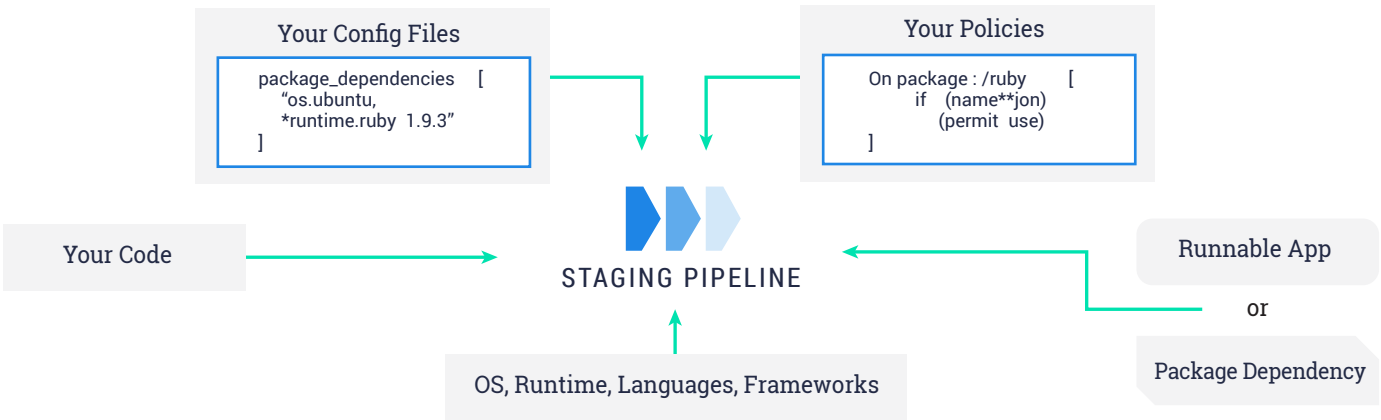
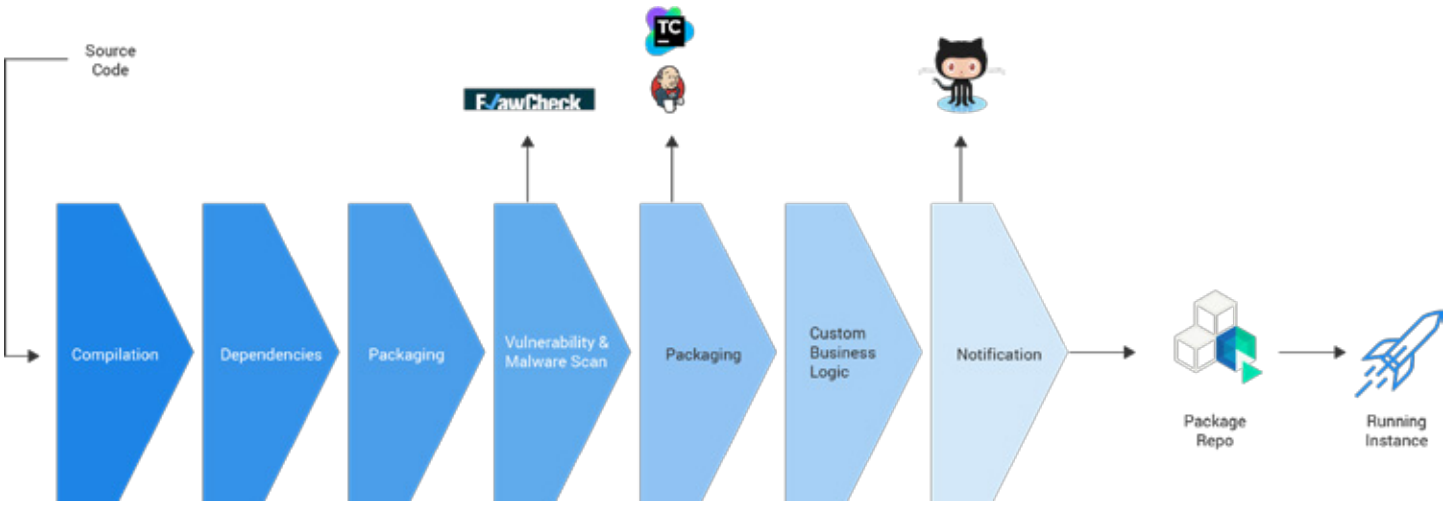


FIGURE 6 | Staging Pipelines



STAGING PIPELINES

A sequence of ordered Stagers forms a Staging Pipeline. The composition of Staging Pipelines can be controlled through policy by different groups of your organization, and as a result control the ingestion of packages into the Apcera platform. The main goal of the Staging Pipelines is to control the quality and continuity of what is ingested into the cluster. Staging Pipelines can also alter packages, such as installing dependencies like rubygems or npm packages.

For instance, the following diagram shows a Staging Pipeline that consists of several stagers, each of which performs some task in the staging process—dependency resolution, packaging, compilation, testing, vulnerability scan, approval, notification and so forth—to create a runnable package.

CONTAINER ISOLATION CONTEXT

An application deployed on Apcera runs within its own self-contained environment known as the Container Isolation Context. The Container Isolation Context is a lightweight and secure Linux container. The Container Isolation Context allows individual jobs to be isolated from other jobs and Apcera components running in the cluster. Isolation includes CPU, memory, file system and visibility on the network.

By default, there is no outside access to or from a Container Isolation Context. Routes are added to a Container Isolation Context to allow ingress traffic, but the container cannot initiate a connection to the outside of the cluster. The application must be paired with Service Bindings to allow egress from the Container Isolation Context.

Container Isolation Context provides workload security, portability, fault tolerance and automation. Many applications can run in a Container Isolation Context inside a single physical or virtual server without visibility into the others' processes, files and network. Each Container Isolation Context can provide single services, such as a web server or a database, or multiple services such as an operating system.

SERVICE PROVIDERS

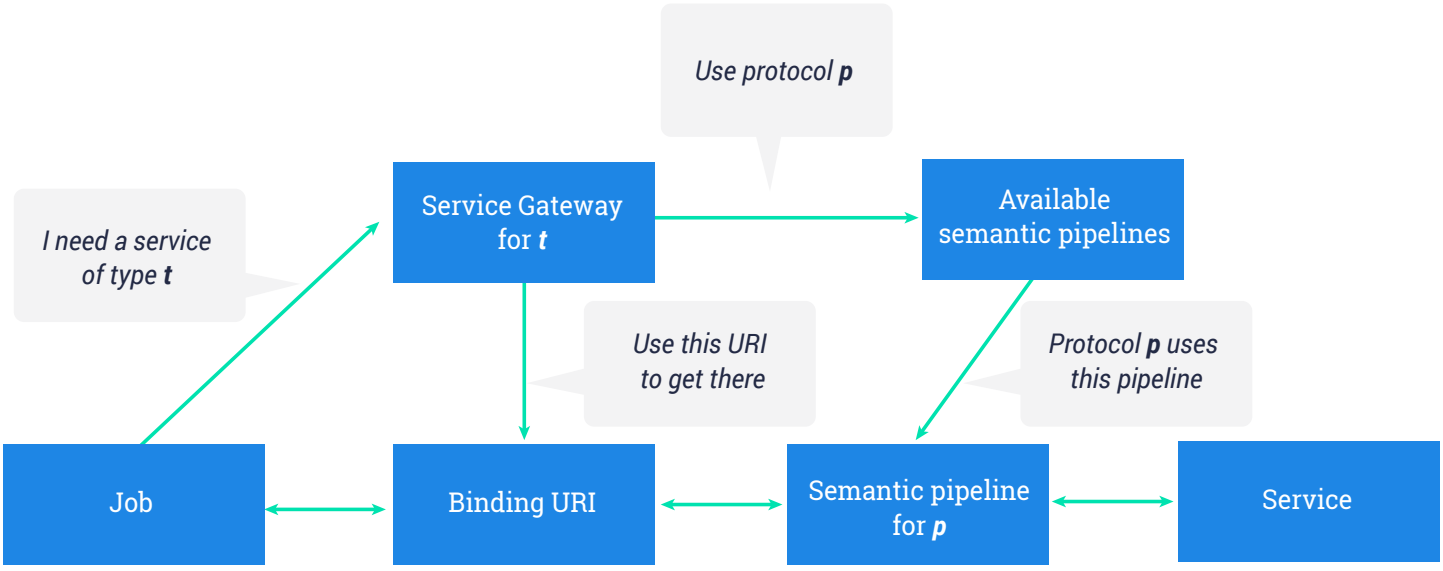
A Service Provider is any backend that provides data over an API, such as HTTP or database protocols. Examples of Service Providers include a database, a queue or caching system, or an account on a third-party SaaS provider including MySQL, PostgreSQL, Redis, NFS, and Memcached. Service providers can be within or outside the Apcera cluster. Enterprise administrators usually designate external resources as Service Providers. Once registered into Apcera, the platform maintains a registry of all Service Providers for the applications to use.

SERVICE GATEWAYS AND SERVICE BINDING

Service Gateways are jobs that manage service providers and configure connections between service providers and other Apcera jobs. Service Gateways ensure that the communication is secure and conforms to your policies.

Every service that a job can connect with has a type and each type has an associated Service Gateway. Service Gateways manage service providers and configure connections, Service Bindings, between services and other Apcera jobs. Apcera includes several built-in Service Gateways, including MySQL, Postgres, Memcache, and Redis. Apcera provides an API to enable you to write Service Gateways to add new types of services to Apcera HCOS.

FIGURE 7 | Service Gateways and Service Binding



The job/app needs to connect with a service (perhaps a database). It passes the service type to the appropriate service gateway, which determines the desired protocol. If, as in this case, Apcera has a semantic pipeline for that protocol, the service gateway deploys it and returns a URI (in this case, the address of the semantic pipeline as a proxy for the web address of the actual service).

FIGURE 8 | Semantic Pipeline

SEMANTIC PIPELINE

In addition to controlling access, addressing, and discovery of network resources, for common protocols, Apcera HCOS provides a framework called a Semantic Pipeline to gain additional insight and control behavior within communication channels in real time. For specific service providers, Apcera does not allow apps and services to communicate directly with each other, but instead manages the communication using Semantic Pipelines. The Semantic Pipeline mediates the connection between the app and the service.

The Semantic Pipeline understands the protocol. It ensures that all communications between the job and the service follow your policies. The Semantic Pipeline performs credential re-mapping using two sets of credentials: one to communicate with the app and another one to communicate with the service a given app is trying to access—minimizing the risk of hacks or leaks. The credentials are ephemeral which means that even if they leaked, they will not work as they are bound to the specific network connection between a specific app and the service in question. The risk that another app could be using leaked credentials is zero.

SERVICE CREDENTIALING

A job can connect to a service by requesting a binding to the service. The binding information provided to the job has credentials to connect to the service. The credentials are designed to be ephemeral, bound to a particular channel and used only by the instance of an application for which the binding was created. So two different instances of an application can share connection to a database, but their credentials are not interchangeable.

CREDENTIALS BINDING URIS

Apcera uses standard URI addresses for services. These URIs are stored as environmental variables so they are always available to any application that needs to know the location of a given service. With this approach, the connectivity information is given to an application rather than built into it. If a DB has moved, the Apcera system will simply restart the application and tell it where the new DB location is. Developers do not have to hard-code that into their applications.

LOGGING

Writing to an app's standard out (stdout) or standard error (stderr) produces a log entry in the format defined by the Syslog protocol. Apcera relies on external Syslog-compatible logging services, for example, Splunk or Papertrail, for long-term storage and management of information logged by the system. You can attach a drain to any app. This sends all log messages from that app to the logging system you specify.

SECURITY

In addition to credential remapping, Apcera protects its internal communications by using the public key infrastructure to ensure that intruders cannot spoof messages. The high-performance NATS messaging system makes it possible to do this efficiently.

Users communicating with Apcera from the outside use an authentication process based on the OAuth2 standard.

NETWORKING

The Instance Manager assigns containers to a dedicated virtual network interface. Traffic to and from the Container Isolation Context is forwarded using network address translation (NAT).

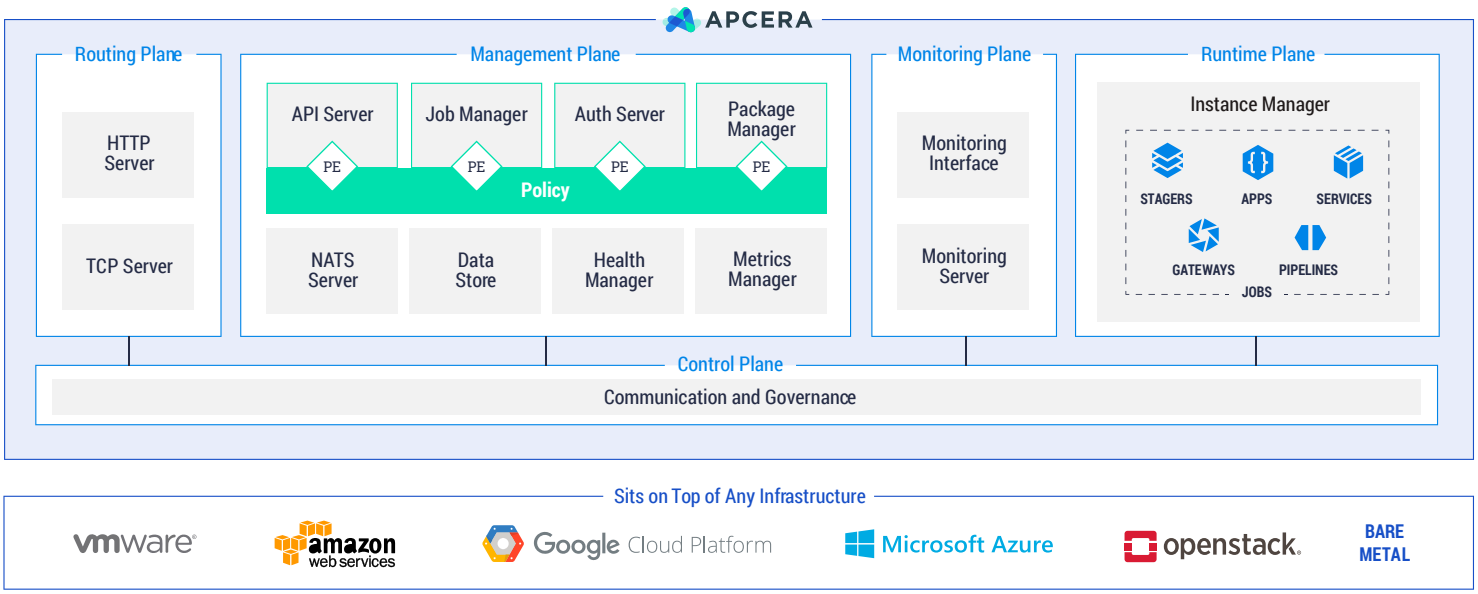
Apcera's NATS-based networking model allows the platform to support:

- Multi-host networking
- Workload firewall
- QoS
- Thousands of containers on a single host
- Network repatching without rolling restarts
- Live network configuration changes without application restarts

CHAPTER 4
HOW APPLICATIONS ARE STAGED AND DEPLOYED

The following series of diagrams provide a high-level, end-to-end overview of how an application is staged and deployed. If you have written a node.js application that allows users to view the contents of your PostgreSQL database via HTTP requests, the database might contain information about the products you sell.

FIGURE 9 | Apcera Architecture



PLATFORM AUTHENTICATION

You begin by authenticating to Apcera using the APC command line tool. APC communicates via HTTPS to the Nginx router, which terminates the connection and forwards the request to the API server over HTTP. HTTP communicates via the NATS high speed bus using encrypted messages to the Authentication server, which authenticates the request.

APPLICATION STAGING

After that, you can begin uploading your application code to Apcera's platform. Apcera automatically selects among the packages it has available to find a combination of runtime and operating system (OS) that meets your code's needs and respects policies that you have established for which code can use which packages. Optionally, you can pass along a manifest file that specifies which versions of node.js your code needs or which operating system you want to use. The staging pipeline component combines the code with the selected runtime and OS to create a package that is ready to run.

APPLICATION DEPLOYMENT AND MONITORING

At this point the Instance Managers receive a request from the Job Manager to start the application. The Instance Managers that have adequate resources to host the application start a bidding process

to respond to the request. The winning Instance Manager communicates back to the Job Manager informing it that the application is now running. The Job Manager instructs the Health Manager to monitor the application and to keep track of the status.

SERVICE BINDING AND SEMANTIC PIPELINES

As part of setting up your system, an administrator designated your PostgreSQL database server as a provider and added it to the Apcera registry of providers. When you ask Apcera to run your app, you also ask it to use that provider to create a service and bind it to your app. Apcera does so and creates a semantic pipeline to run alongside the app and monitor its communication with the server. It also provides your app with credentials for communicating with the PostgreSQL database. The semantic pipeline enforces your policies about which database services your app can access

DATA TRAFFIC ROUTING

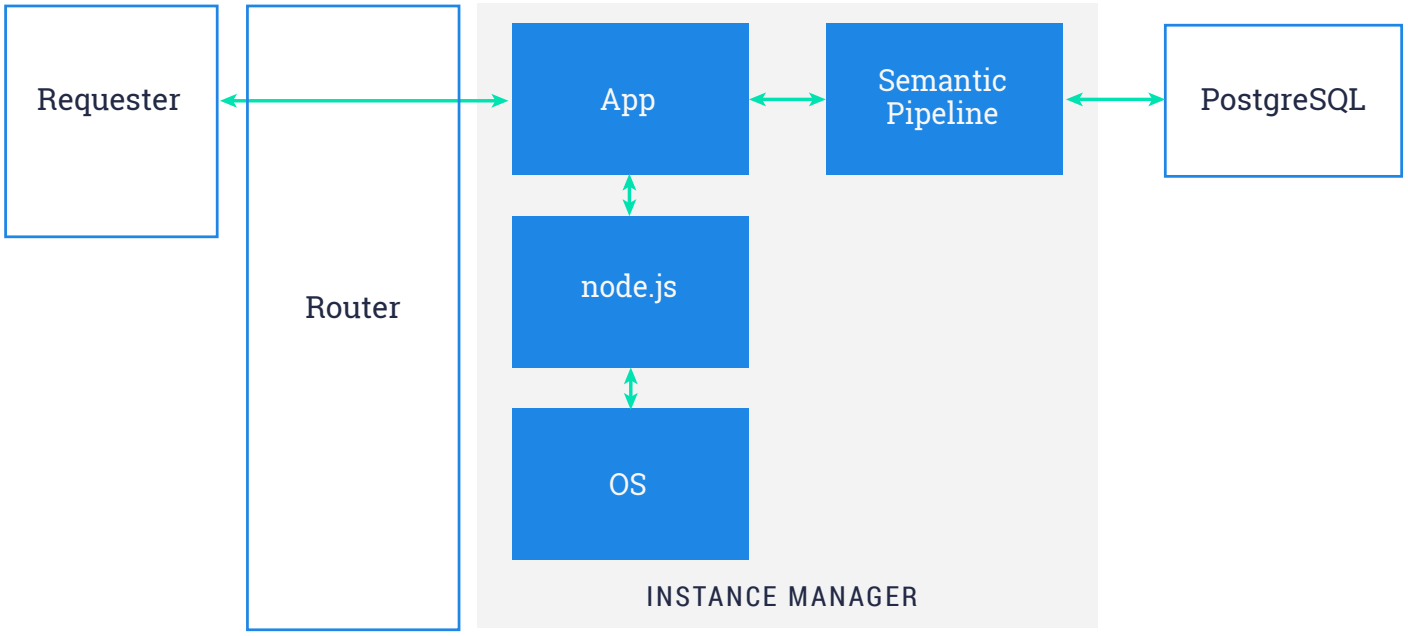
In order for your app to accept outside requests, subject to your policies, you assign it a route that is, a URL and port to listen on. The Instance Manager that runs your job turns that into a URL and port and configures the Nginx router to ensure that requests sent to your job actually reach the correct instance.

When HTTP requests arrive for your app, they arrive at an instance of your job which then communicates via its associated semantic pipeline with the PostgreSQL server and returns a result back through the Nginx router to the requester.

SSH INTO THE CONTAINER

To connect to a container hosting an application via SSH, Apcera establishes a secure WebSocket to the Nginx router which terminates the connection and forwards the request to the API server over HTTP. HTTP communicates via the NATS high speed bus using encrypted messages to the Instance Manager, which establishes the communication with the container hosting the application.

FIGURE 10 | Instance Manager



ABOUT APCERA

Apcera is the market leading enterprise-grade container management platform — driven by security and policy — that gives IT leaders the confidence and control to drive innovation and move faster, securely. Built for cloud-native as well as legacy applications, Apcera lets IT teams containerize, deploy, orchestrate and govern a vast range of workloads across on-premises, cloud and hybrid environments. Fully integrated and completely turnkey, only Apcera offers industry-leading agility and time-to-value without sacrificing security or control.

Apcera enables key enterprise use cases including deploying Docker in production securely and at scale, legacy application modernization and hybrid cloud mobility. Global 2000 companies use Apcera to lower CapEx and OpEx, improve time to market and reduce risk.

Apcera is headquartered in San Francisco. For more information, visit https://www.apcera.com/, read the company's blog or follow on Twitter: @apcera.