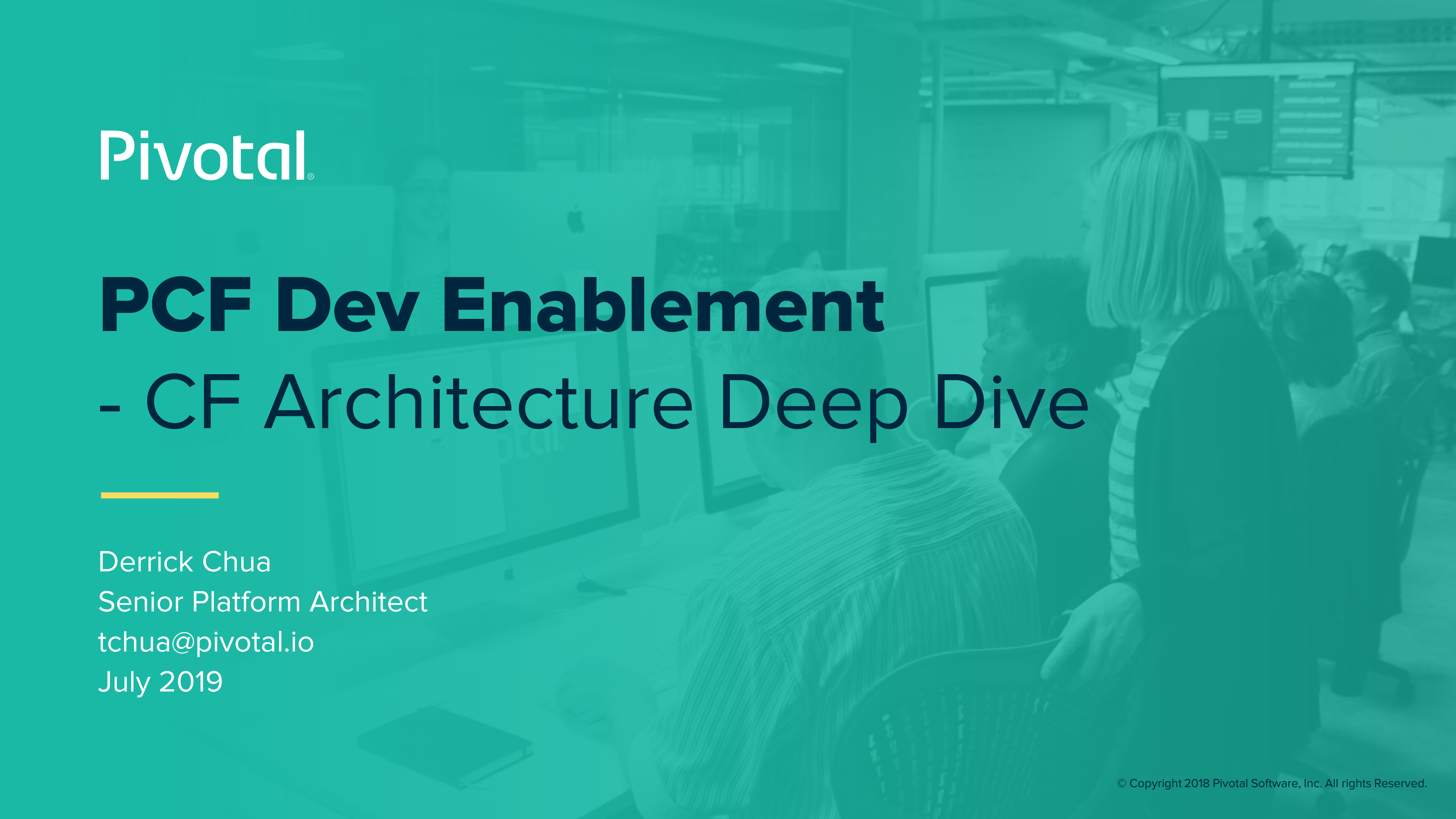


Pivotal

PCF Dev Enablement
- CF Architecture Deep Dive

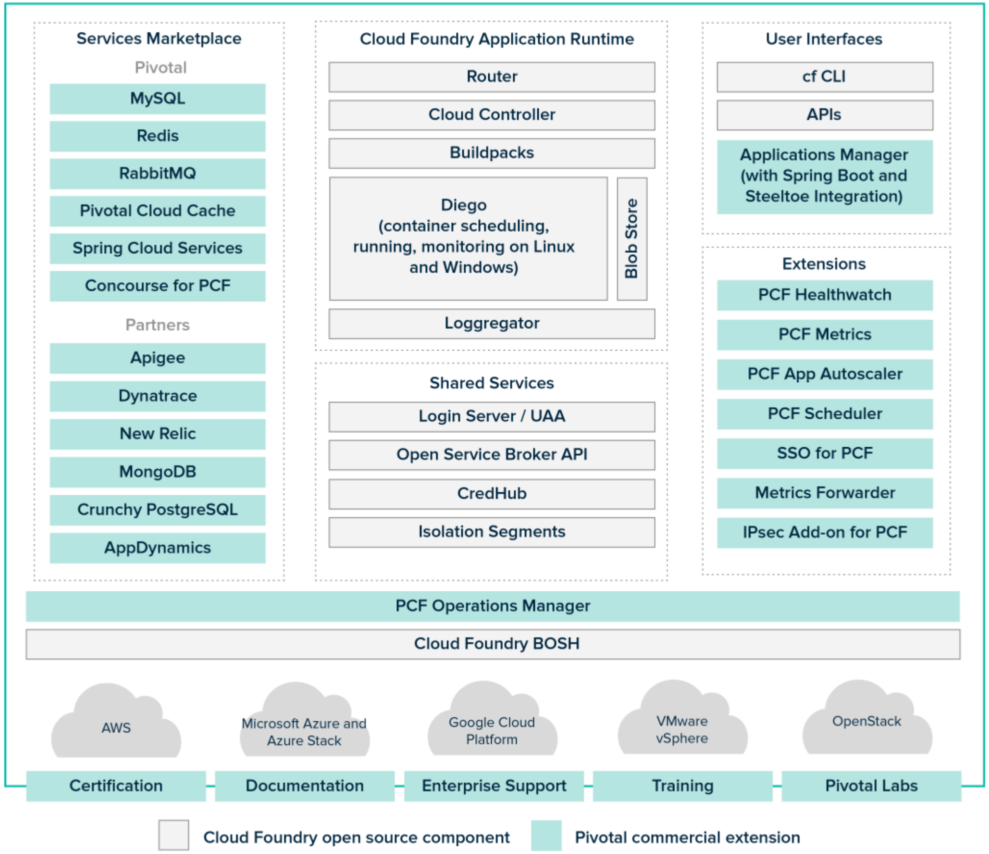
Derrick Chua
Senior Platform Architect
tchua@pivotal.io

July 2019

© Copyright 2018 Pivotal Software, Inc. All rights Reserved.

PAS vs Opensource CF

Pivotal Application Service



Services Marketplace

Pivotal

- MySQL
- Redis
- RabbitMQ
- Pivotal Cloud Cache
- Spring Cloud Services
- Concourse for PCF

Partners

- Apigee
- Dynatrace
- New Relic
- MongoDB
- Crunchy PostgreSQL
- AppDynamics

Cloud Foundry Application Runtime

- Router
- Cloud Controller
- Buildpacks
- Diego (container scheduling, running, monitoring on Linux and Windows)
- Blob Store
- Loggregator

Shared Services

- Login Server / UAA
- Open Service Broker API
- CredHub
- Isolation Segments

User Interfaces

- cf CLI
- APIs
- Applications Manager (with Spring Boot and Steeltoe Integration)

Extensions

- PCF Healthwatch
- PCF Metrics
- PCF App Autoscaler
- PCF Scheduler
- SSO for PCF
- Metrics Forwarder
- IPsec Add-on for PCF

PCF Operations Manager

Cloud Foundry BOSH

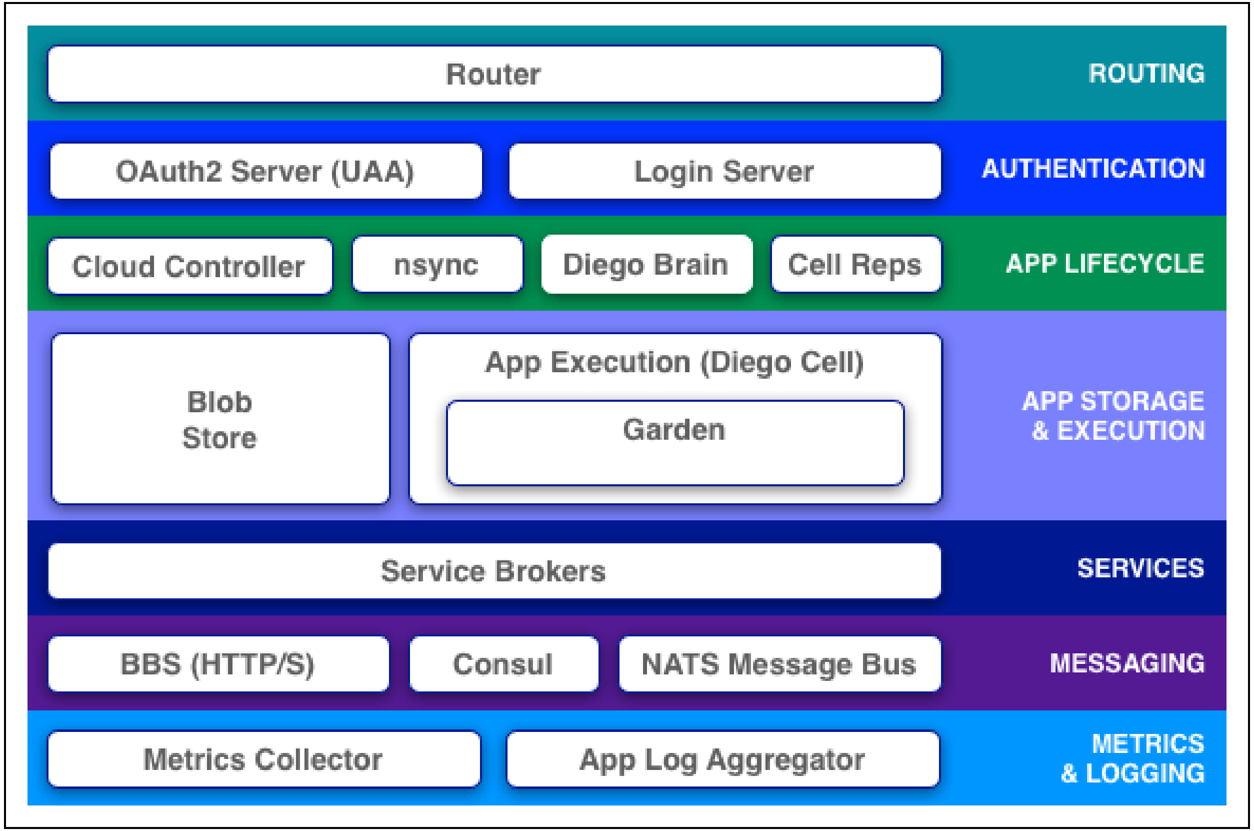
- AWS
- Microsoft Azure and Azure Stack
- Google Cloud Platform
- VMware vSphere
- OpenStack

Certification | Documentation | Enterprise Support | Training | Pivotal Labs

□ Cloud Foundry open source component

🟩 Pivotal commercial extension

CF Runtime Components

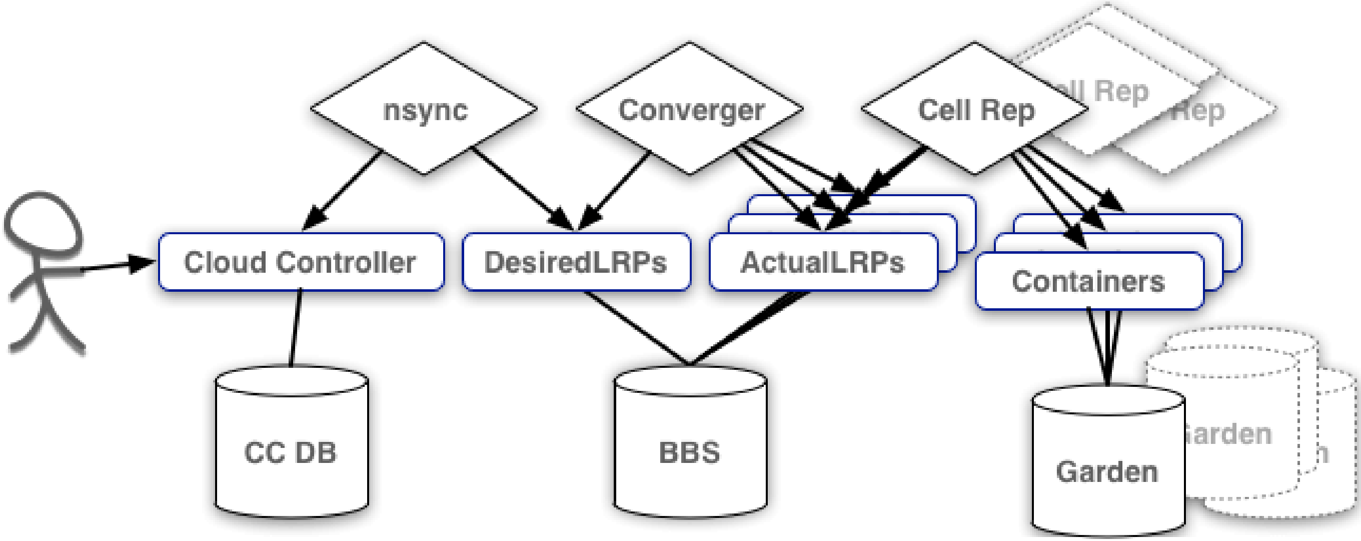


Router	ROUTING
OAuth2 Server (UAA) Login Server	AUTHENTICATION
Cloud Controller nsync Diego Brain Cell Reps	APP LIFECYCLE
Blob Store App Execution (Diego Cell) Garden	APP STORAGE & EXECUTION
Service Brokers	SERVICES
BBS (HTTP/S) Consul NATS Message Bus	MESSAGING
Metrics Collector App Log Aggregator	METRICS & LOGGING

Details at https://docs.pivotal.io/pivotalcf/2-6/concepts/architecture/index.html

nsync, BBS and Cell Reps

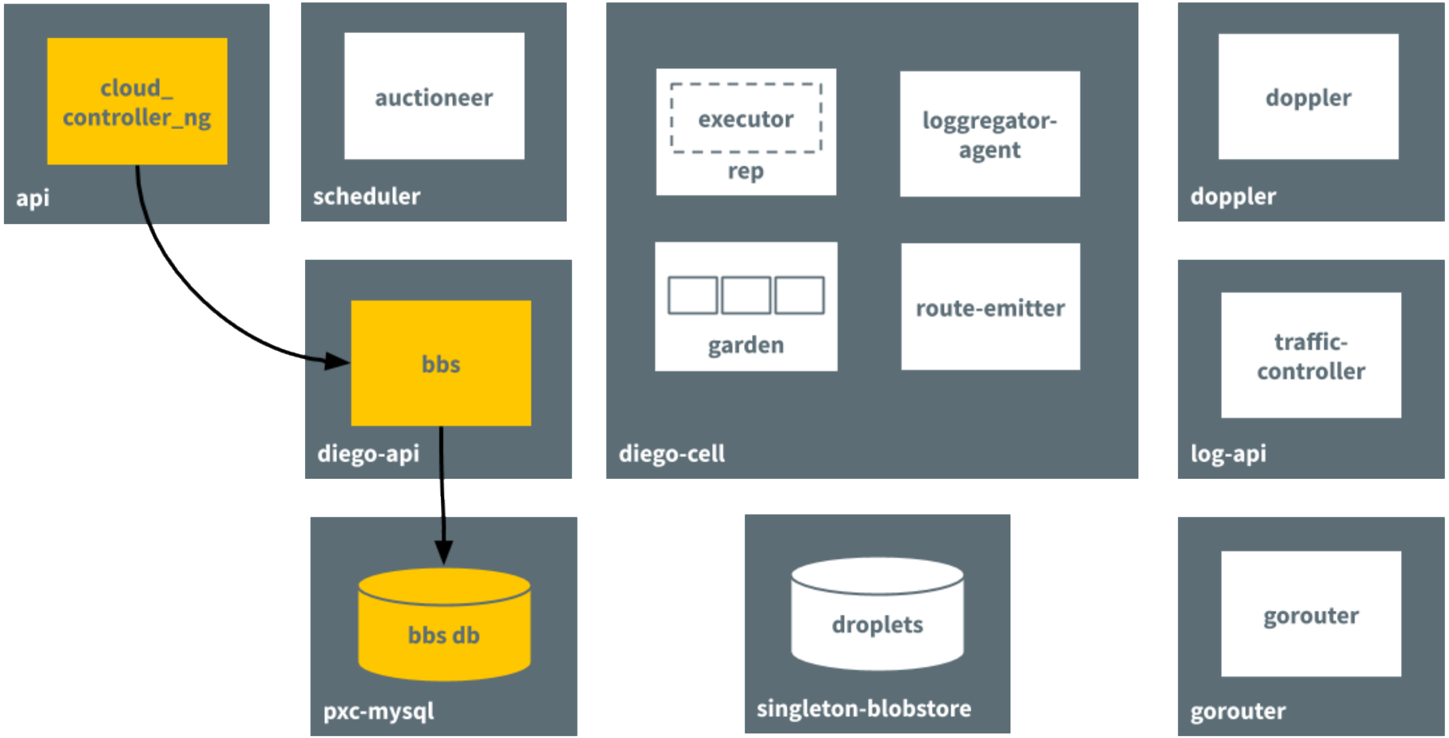
Monitor and reconcile application states



App Monitoring and Syncing with Diego

How Diego runs an app

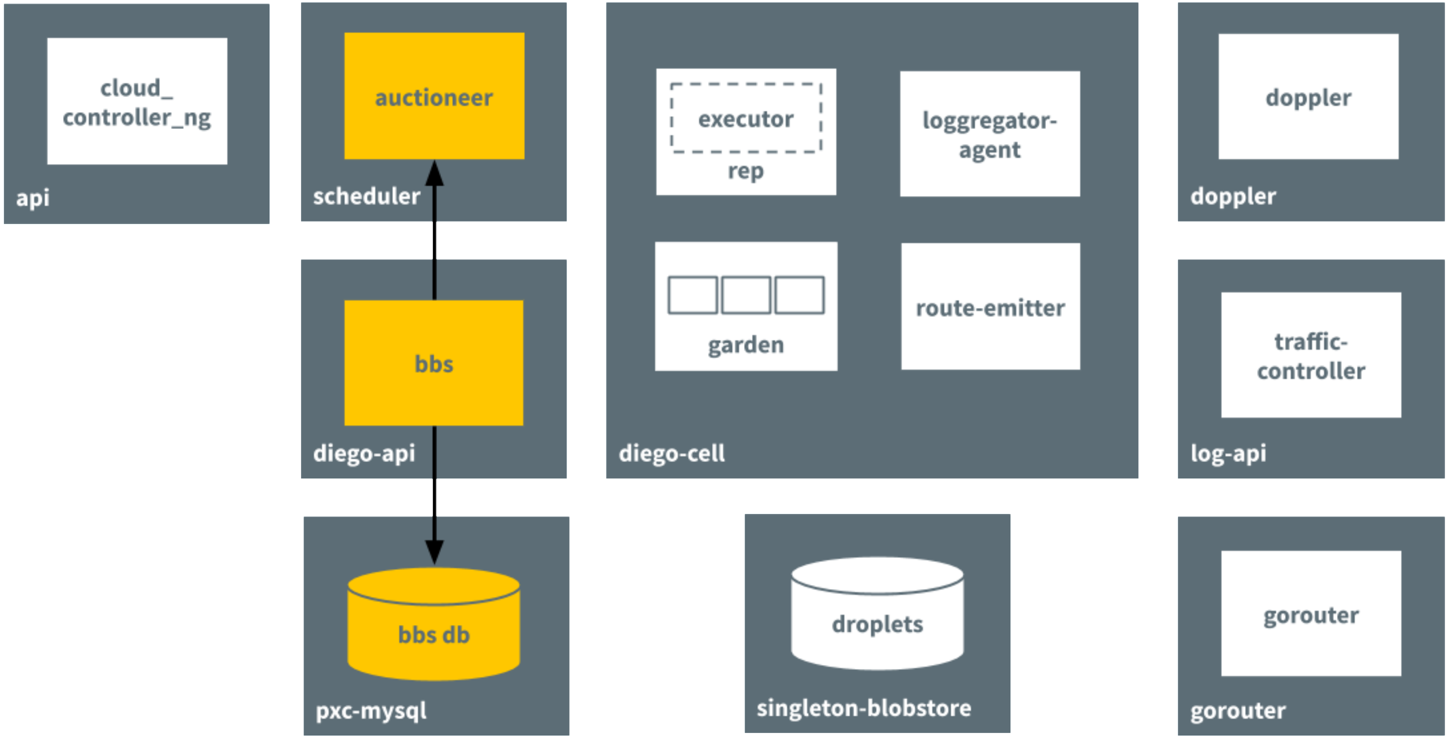
Step 1: Receives request to run an app



Pivotal

How Diego runs an app

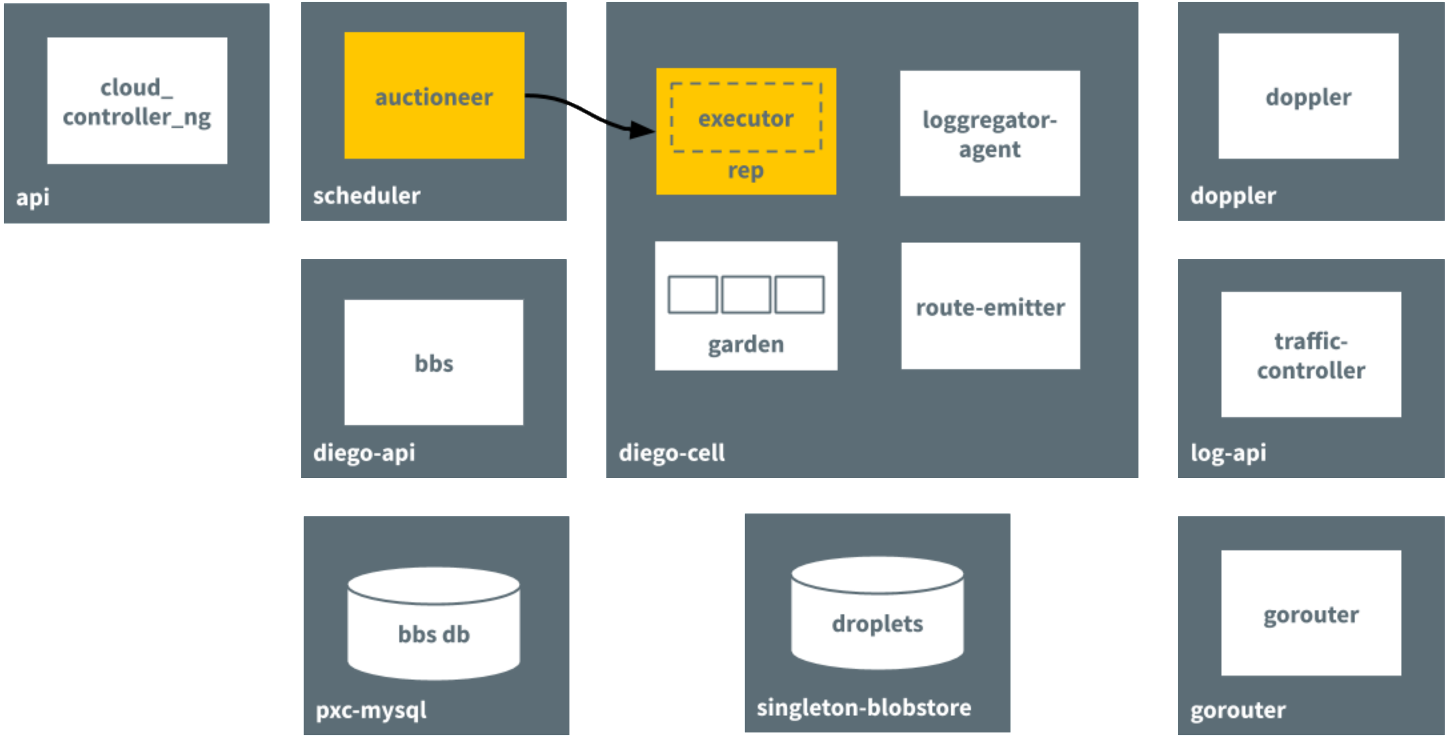
Step 2: Pass request to auctioneer process



Pivotal

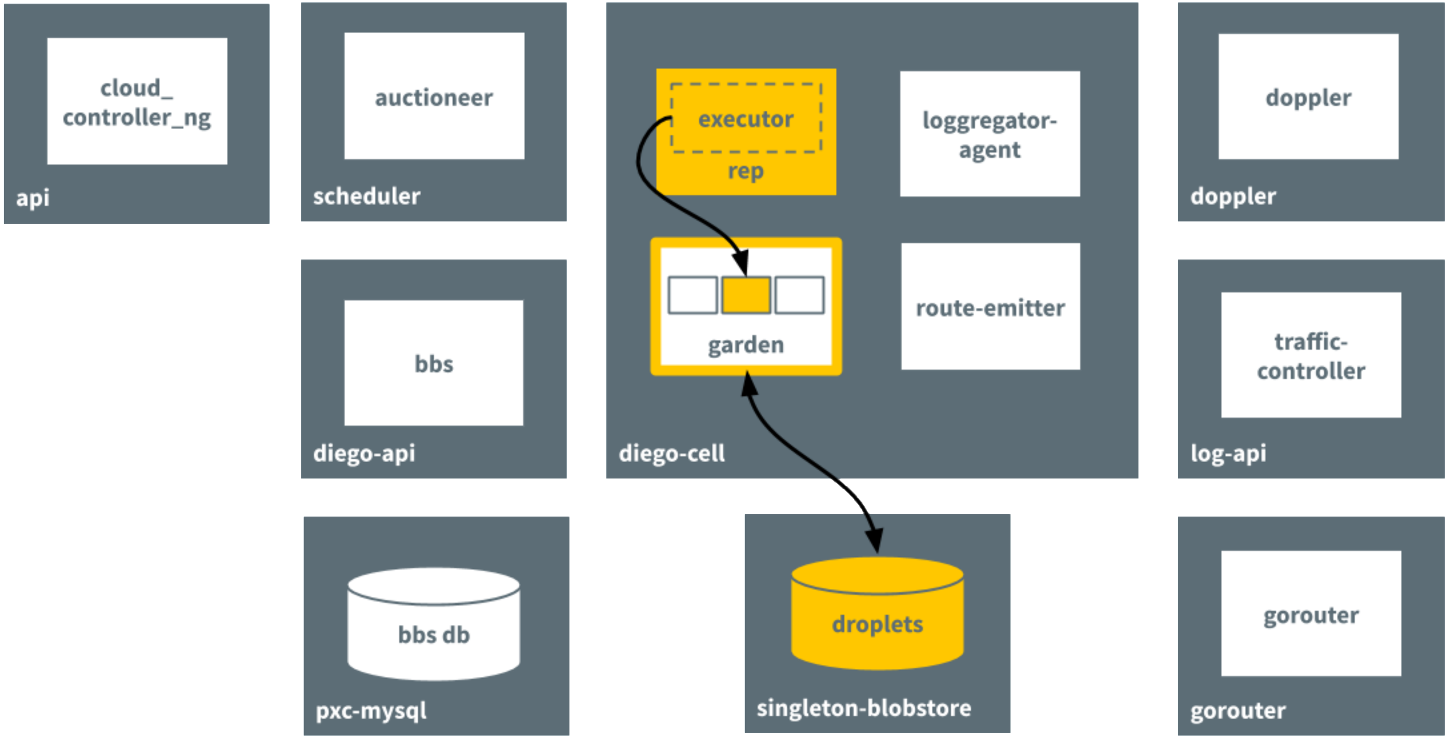
How Diego runs an app

Step 3: Performs auction



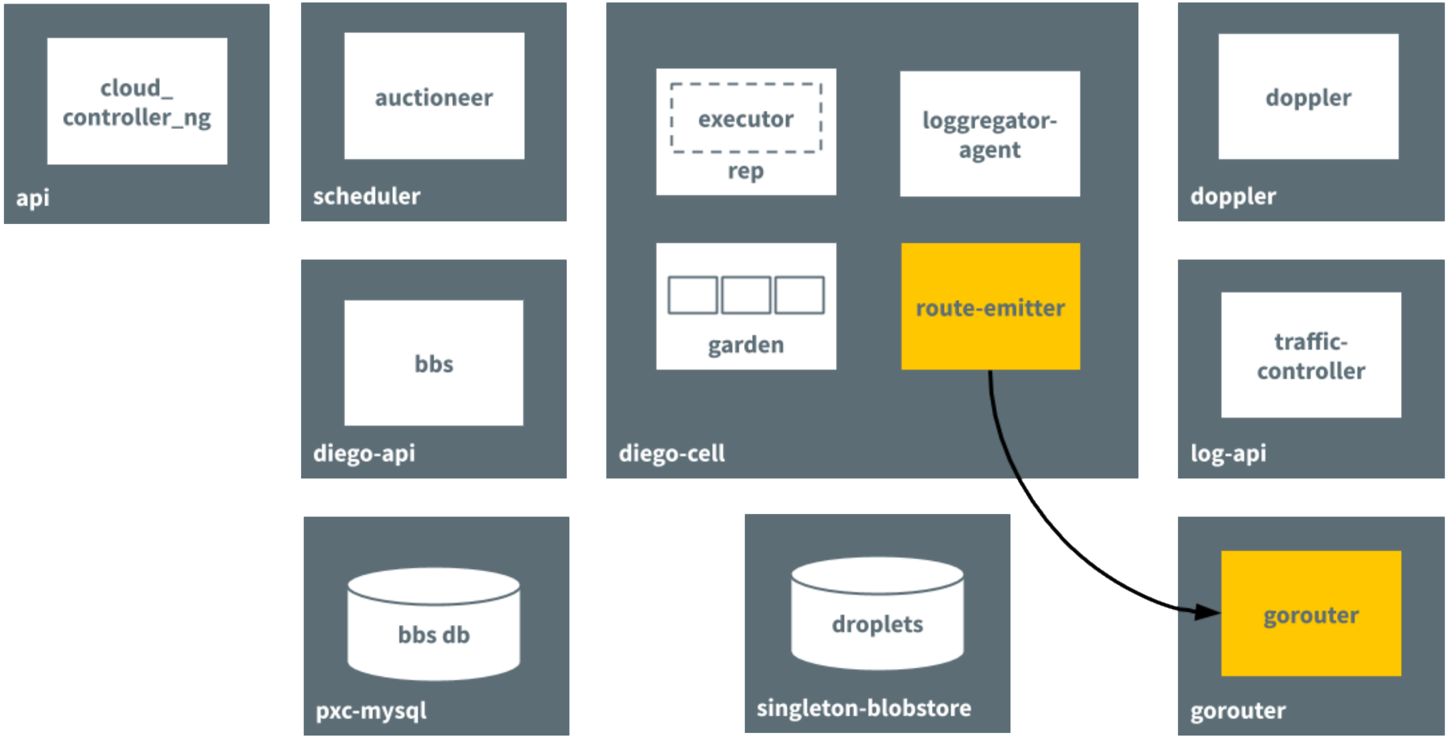
How Diego runs an app

Step 4: Creates container and runs app



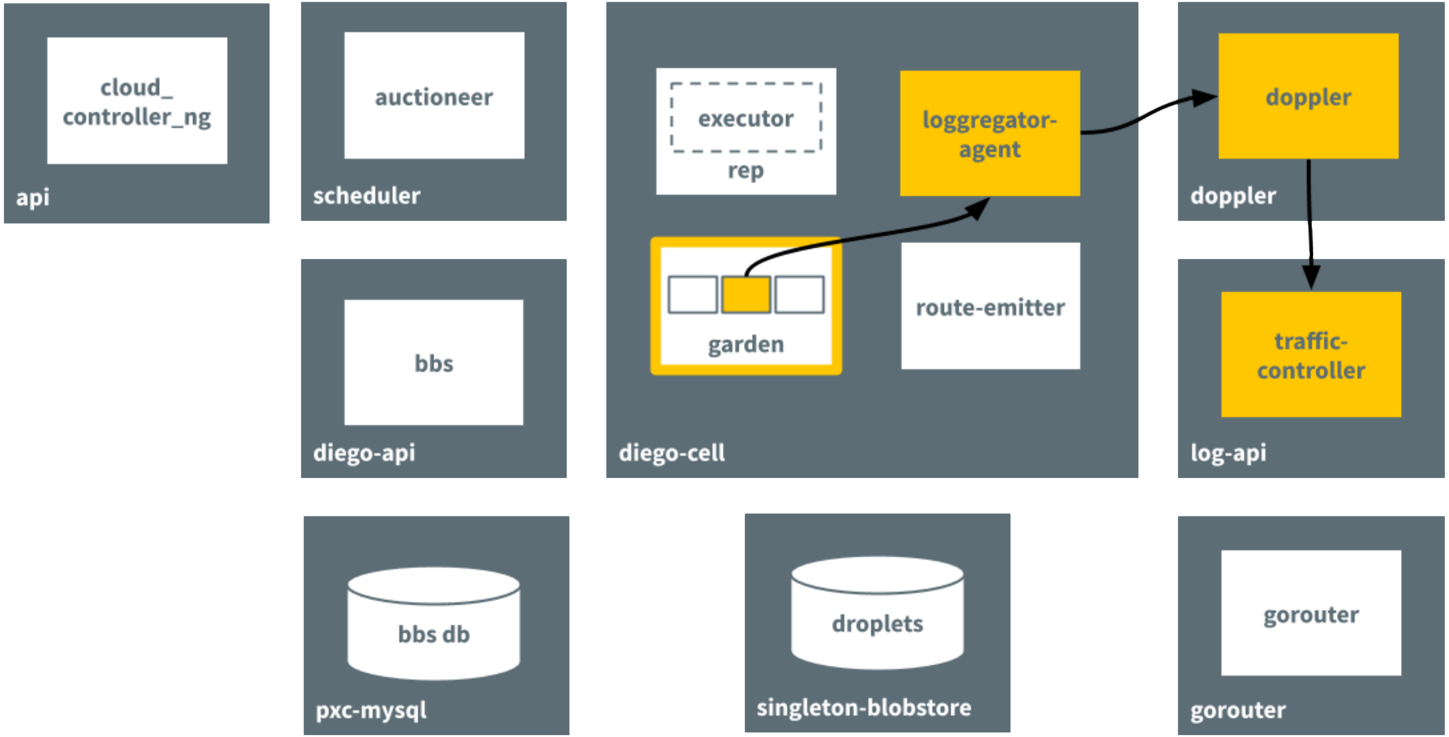
How Diego runs an app

Step 5: Emits routes for app



How Diego runs an app

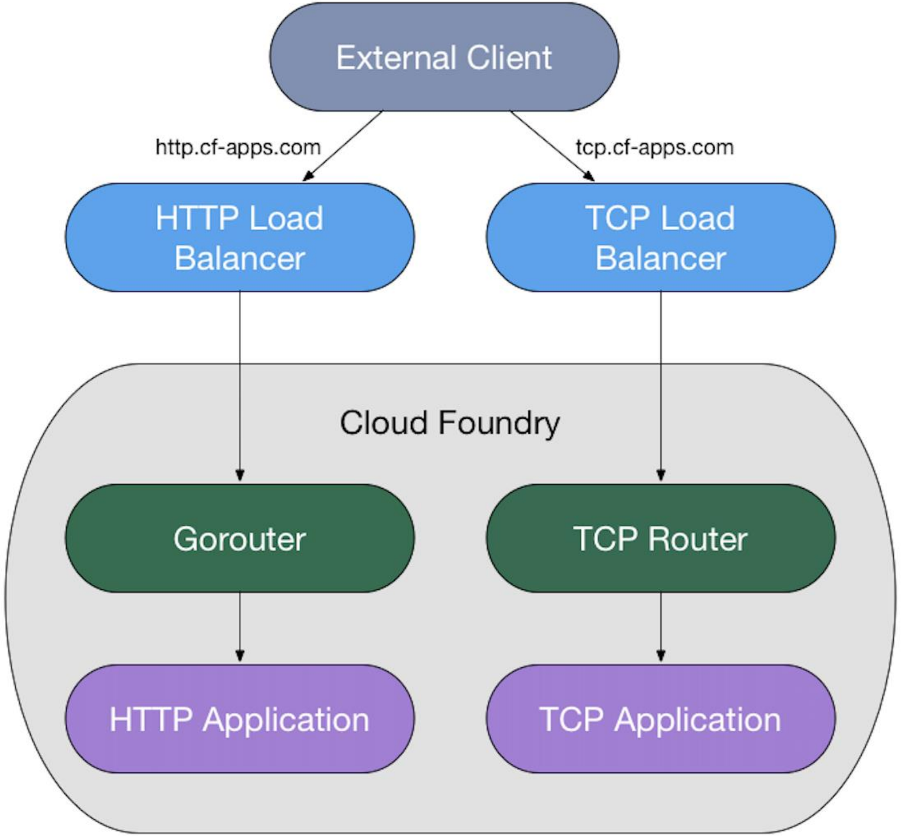
Step 6: Sends logs to loggregator



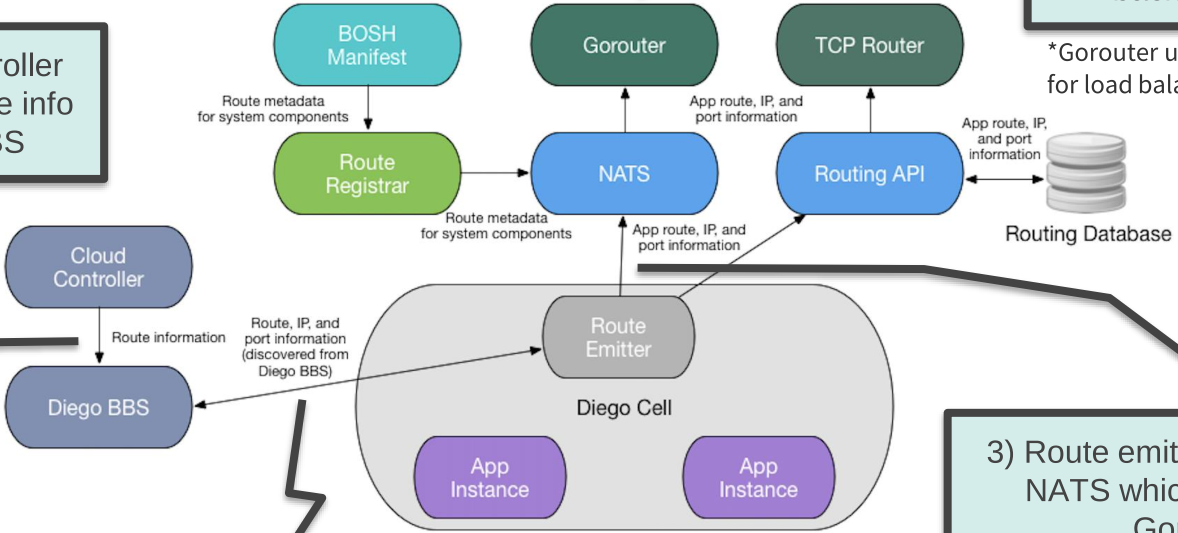
Pivotal

Routing

External client request flow



Maintaining updated routing table (HTTP)


BOSH Manifest
Route metadata for system components
Gorouter
TCP Router
App route, IP, and port information
Route Registrar
NATS
Routing API
App route, IP, and port information
Routing Database
Route metadata for system components
App route, IP, and port information
Cloud Controller
Route information
Route, IP, and port information (discovered from Diego BBS)
Route Emitter
Diego BBS
Diego Cell
App Instance
App Instance

1) Cloud Controller sends app route info to Diego BBS

2) Route emitter queries for app route and backend IP address and port from Diego BBS

3) Route emitter sends info to NATS which forwards to Gorouter

4) Gorouter uses info to map incoming app requests to backend app instances

*Gorouter uses the round-robin algorithm for load balancing

HTTP Routing

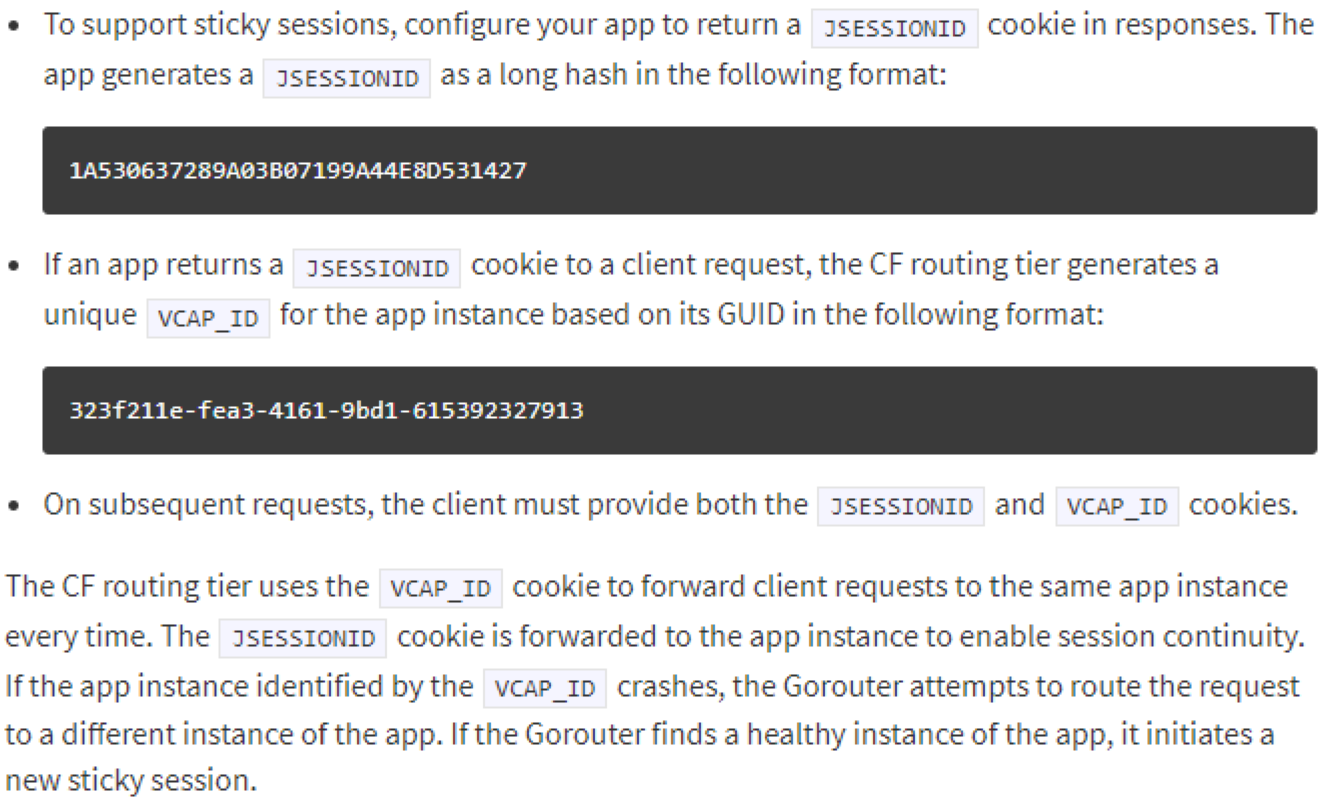
Headers

Header name	Purpose
X-Forwarded-Proto	• Gives the scheme of the HTTP request from the client • HTTP for insecure request, HTTPS for secure request • Multiple values – comma separated list • App should process to reject insecure requests
X-Forwarded-For	• Load balancer IP address
X-B3-TraceId X-B3-SpanId	• Zipkin tracing • Logged to Gorouter logs
X-CF-APP-INSTANCE	• Used to obtain debug data for a specific instance of an app
X-Forwarded-Client-Cert	• For mutual TLS • Used to pass the originating client certificate along the data path to the application • If LB terminates TLS, this header should be stripped to prevent client spoofing

** Gorouter has a limit of 1 MB for HTTP Headers

HTTP Routing

Session affinity (sticky sessions)



- To support sticky sessions, configure your app to return a `JSESSIONID` cookie in responses. The app generates a `JSESSIONID` as a long hash in the following format:

```
1A530637289A03B07199A44E8D531427
```

- If an app returns a `JSESSIONID` cookie to a client request, the CF routing tier generates a unique `VCAP_ID` for the app instance based on its GUID in the following format:

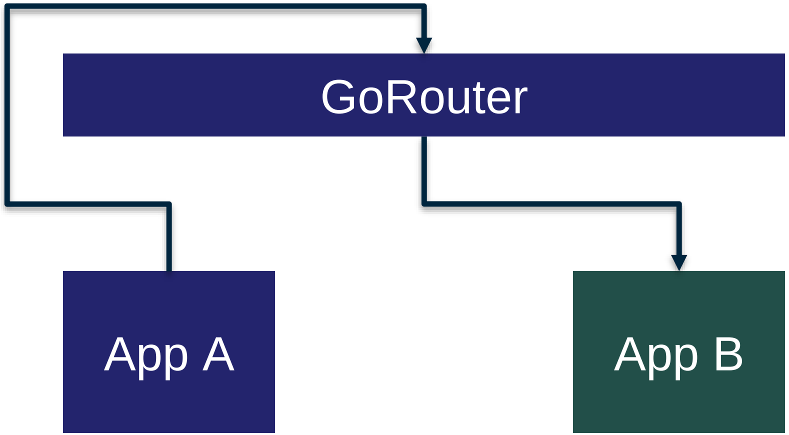
```
323f211e-fea3-4161-9bd1-615392327913
```

- On subsequent requests, the client must provide both the `JSESSIONID` and `VCAP_ID` cookies.

The CF routing tier uses the `VCAP_ID` cookie to forward client requests to the same app instance every time. The `JSESSIONID` cookie is forwarded to the app instance to enable session continuity. If the app instance identified by the `VCAP_ID` crashes, the Gorouter attempts to route the request to a different instance of the app. If the Gorouter finds a healthy instance of the app, it initiates a new sticky session.

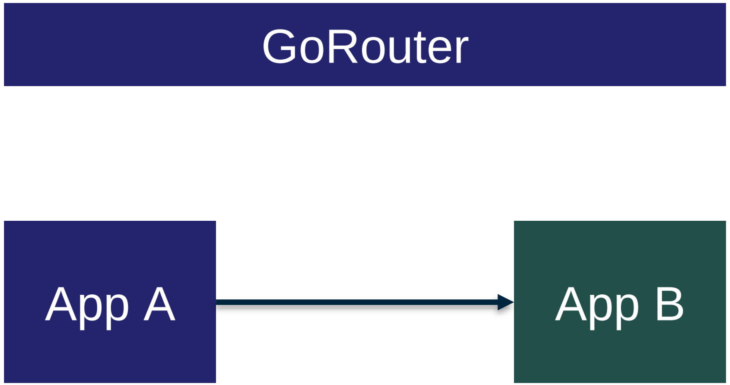
Container to Container Networking

What happens without container to container networking



Container to Container Networking

What happens with container to container networking

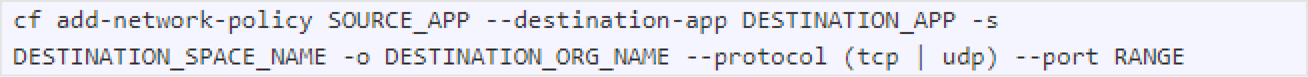


Container to Container Networking

How to...

- By default, each Diego cell in the overlay network is allocated a /24 range that supports 254 containers per cell, one container for each of the usable IP addresses

- Add a network policy to allow one app to talk to another



```
cf add-network-policy SOURCE_APP --destination-app DESTINATION_APP -s
DESTINATION_SPACE_NAME -o DESTINATION_ORG_NAME --protocol (tcp | udp) --port RANGE
```

- With **app service discovery**, apps pushed to Pivotal Application Service (PAS) can establish container-to-container communications through a known route served by internal BOSH DNS
  - Default internal domain is apps.internal
  - PAS apps can reach each other through [APP_NAME].apps.internal

Application Security Groups

- Collection of **egress rules** that specify the protocols, ports, and IP address ranges where app or task instances send traffic

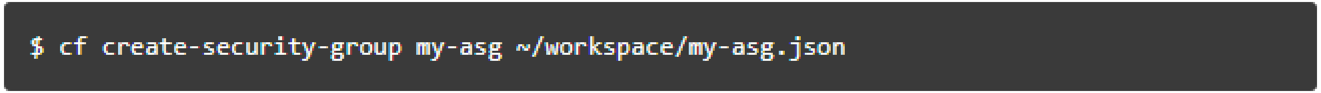
- Define **allow** rules, and their order of evaluation is unimportant when multiple ASGs apply to the same space or deployment

- Administrators can define **a staging ASG** for app and task staging, and a **running ASG** for app and task runtime
  - Staging ASG is typically less restrictive and is used to pull resources required during staging

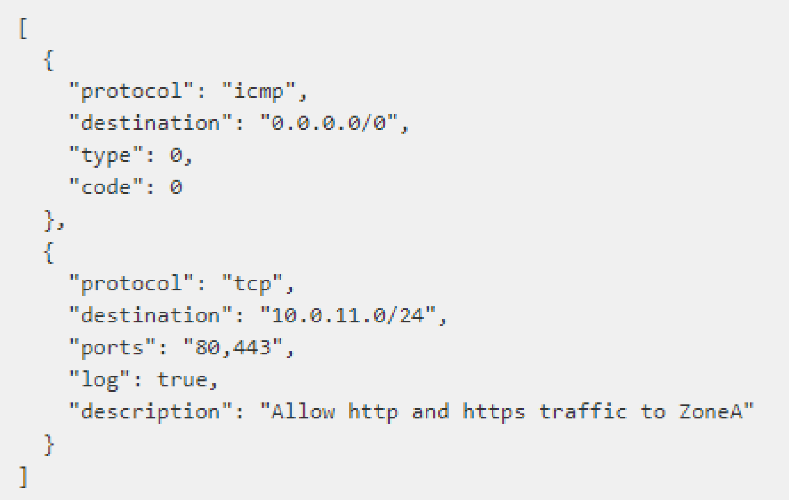
- Administrators can assign **platform-wide ASGs** that apply to all app and task instances for the entire deployment, or **space-scoped ASGs** that apply only to apps and tasks in a particular space

Application Security Groups

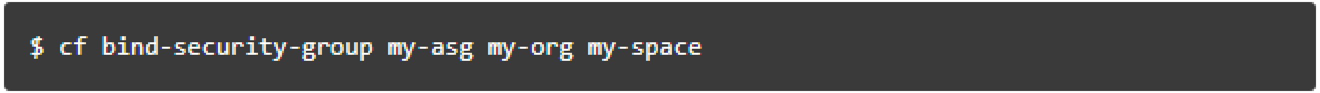
Creating and binding ASGs



```
$ cf create-security-group my-asg ~/workspace/my-asg.json
```



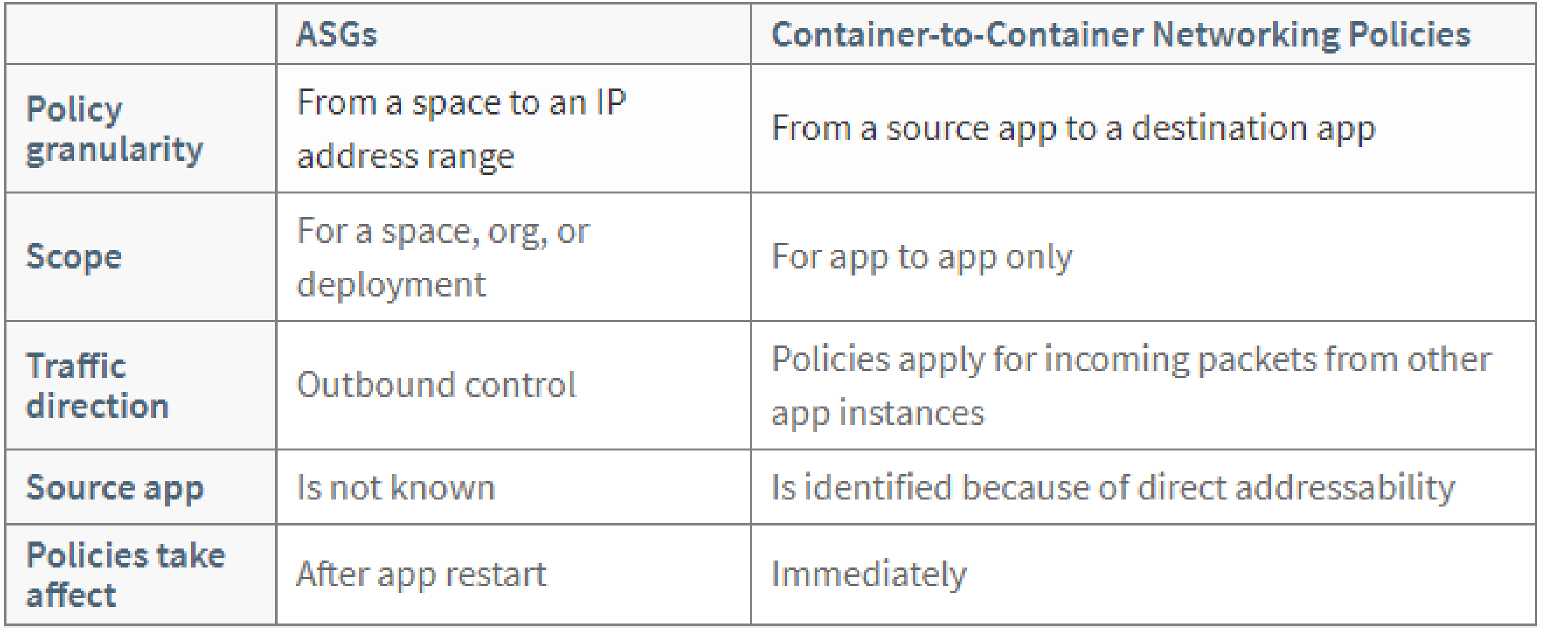
```
[
  {
    "protocol": "icmp",
    "destination": "0.0.0.0/0",
    "type": 0,
    "code": 0
  },
  {
    "protocol": "tcp",
    "destination": "10.0.11.0/24",
    "ports": "80,443",
    "log": true,
    "description": "Allow http and https traffic to ZoneA"
  }
]
```



```
$ cf bind-security-group my-asg my-org my-space
```

Details at https://docs.pivotal.io/pivotalcf/2-6/concepts/asg.html

Container to Container Networking vs ASGs



	ASGs	Container-to-Container Networking Policies
Policy granularity	From a space to an IP address range	From a source app to a destination app
Scope	For a space, org, or deployment	For app to app only
Traffic direction	Outbound control	Policies apply for incoming packets from other app instances
Source app	Is not known	Is identified because of direct addressability
Policies take affect	After app restart	Immediately

The (Near) Future

Pivotal®

Transforming How The World Builds Software

© Copyright 2018 Pivotal Software, Inc. All rights Reserved.