Contents

Chapter 1

Managing a Graphistry Deployment

Welcome to Graphistry!

Quick start

The fastest way to install, admininster, and use Graphistry is to quick launch Graphistry from the AWS Marketplace (see walkthrough tutorial & videos). AWS Marketplace launches a Graphistry instance in your private cloud and runs with zero additional configuration necessary.

Advanced administration

Graphistry supports advanced command-line administration via standard docker-compose `.yml` / `.env` files and `nginx` / `caddy` configuration.

The `graphistry-cli` repository contains * Documentation for operating the Graphistry Docker container (install, configure, start/stop, & debug) * Documentation for configuring the software: `nginx`, connectors, and ontology

Manual Install for Nvidia Environments, Including AWS

Install Graphistry container

If `nvidia` is already your `docker info | grep Default` runtime:

```
############ Install & Launch
wget -O release.tar.gz "https://..."
tar -xvvf release.tar.gz
docker load -i containers.tar
docker-compose up -d
```

Docker: Launch & Configure Nvidia for Docker

AWS Nvidia Ubuntu Deep Learning AMIs have everything except you need to enable the default docker runtime:

```
############ Environment
$ docker info | grep Default    ### => runc
$ sudo vim /etc/docker/daemon.json
{
    "default-runtime": "nvidia",
    "runtimes": {
        "nvidia": {
            "path": "nvidia-container-runtime",
            "runtimeArgs": []
        }
    }
}
$ sudo systemctl restart docker ### without, may need `docker system prune -a && docker syst
$ docker info | grep Default    ### => nvidia
```

The Graphistry environnment depends soley on Nvidia RAPIDS and Nvidia Docker via `Docker Compose 3`, and ships with all other dependencies built in.

Top Commands

Install	`docker load -i containers.tar`	Install the `containers.tar` Graphistry release from the current folder. You may need to first run `tar -xvvf my-graphistry-release.tar.gz`.
Start (interactive)	`docker-compose up`	Starts Graphistry, close with ctrl-c
Start (daemon)	`docker-compose up -d`	Starts Graphistry as background process
Stop	`docker-compose stop`	Stops Graphistry
Restart	`docker restart <CONTAINER>`	
Status	`docker-compose ps`, `docker ps`, and `docker status`	Status: Uptime, healthchecks, …

API Key	docker-compose exec streamgo-vgraph-etl curl “http://0.0.0.0:8080/api/internal/provision?text=MYUSERNAME”	Generates API key for a developer or notebook user
Logs	`docker logs <CONTAINER>` (or `docker exec -it <CONTAINER>` followed by `cd /var/log`)	Ex: Watch all logs, starting with the 20 most recent lines: `docker-compose logs -f -t --tail=20`

Reset	`docker-compose down -v && docker-compose up`	Stop Graphistry, remove all internal state (ex: user accounts), and start fresh.

Contents

Chapter 2

Instance & Environment Setup

1. Prerequisites

- AWS Marketplace: Quota for GPU (P3.2+) in your region; ignore everything else below
- Graphistry Docker container
- Linux with `nvidia-docker-2`, `docker-compose`, and `CUDA 9.2`. Ubuntu 16.04 cloud users can use a Graphistry provided environment bootstrapping script.
- NVidia GPU: K80 or later. Recommended G3+ on AWS and NC Series on Azure.
- Browser with Chrome or Firefox

For further information, see Recommended Deployment Configurations: Client, Server Software, Server Hardware.

2. Instance Provisioning

AWS Marketplace (Recommended)

- Use any of the recommended instance types: P3.2+

AWS BYOL - From a new Nvidia AMI

- Launch a base Nvidia Deep Learning Ubuntu AMI on a `p3.*`

- Use S3AllAccess permissions, and override default parameters for: 200GB disk
- Enable SSH/HTTP/HTTPS in the security groups
- SSH as `ubuntu@[your ami]`
- Set `nvidia` as the default docker run-time:

  ```
  $ docker info | grep Default    ### => runc
  $ sudo vim /etc/docker/daemon.json
  {
  "default-runtime": "nvidia",
  "runtimes": {
      "nvidia": {
          "path": "nvidia-container-runtime",
          "runtimeArgs": []
      }
  }
  }
  $ sudo systemctl restart docker ### without, may need `docker system prune -a && docker
  $ docker info | grep Default    ### => nvidia
  ```

- Follow `docker load` instructions up top.

AWS BYOL - From a base Linux AMI

- Launch an official AWS Ubuntu 16.04 LTS AMI using a `g3+`or `p*` GPU instance.
- Use S3AllAccess permissions, and override default parameters for: 200GB disk
- Enable SSH/HTTP/HTTPS in the security groups
- SSH as `ubuntu@[your ami]`, `centos@`, or `ec2-user@`.

Proceed to the OS-specific instructions below.

For further information, see full AWS installation instructions.

Azure

- Launch an Ubuntu 16.04 LTS Virtual Machine with an `NC*` GPU compute SKU, e.g., NC6 (hdd)
- Enable SSH/HTTP/HTTPS
- Check to make sure GPU is attached

```
$ lspci -vnn | grep VGA -A 12
0000:00:08.0 VGA compatible controller [0300]: Microsoft Corporation Hyper-V virtual VGA [14
    Flags: bus master, fast devsel, latency 0, IRQ 11
```

```
    Memory at f8000000 (32-bit, non-prefetchable) [size=64M]
    [virtual] Expansion ROM at 000c0000 [disabled] [size=128K]
    Kernel driver in use: hyperv_fb
    Kernel modules: hyperv_fb

5dc5:00:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev a
    Subsystem: NVIDIA Corporation GK210GL [Tesla K80] [10de:106c]
    Flags: bus master, fast devsel, latency 0, IRQ 24, NUMA node 0
    Memory at 21000000 (32-bit, non-prefetchable) [size=16M]
    Memory at 1000000000 (64-bit, prefetchable) [size=16G]
    Memory at 1400000000 (64-bit, prefetchable) [size=32M]
```

Proceed to the OS-specific instructions below.

For further information, see full Azure installation instructions.

On-Premises

See Recommended Deployment Configurations: Client, Server Software, Server Hardware.

Airgapped

Graphistry runs airgapped without any additional configuration. Pleae contact your systems representative for assistance with nvidia-docker-2 environment setup.

3. Linux Dependency Installation

If your environment already has `nvidia-docker-2`, `docker`, `docker-compose`, and `CUDA 9.2`, skip this section.

Ubuntu 16.04 LTS

```
    $ git clone https://github.com/graphistry/graphistry-cli.git
    $ bash graphistry-cli/bootstrap.sh ubuntu-cuda9.2
```

RHEL 7.4 / CentOS 7

Note: Temporarily not supported on AWS/Azure

```
$ sudo yum install -y git
$ git clone https://github.com/graphistry/graphistry-cli.git
$ bash graphistry-cli/bootstrap.sh rhel
```

After

Log off and back in (full restart not required): "`$ exit`", "`$ exit`"

Warning: Skipping this step means `docker` service may not be available

Warning: Skipping this step means Graphistry environment tests will not automatically run

Test environment

These tests run upon exiting the bootstrap. You can invoke them manually at any time:

```
$ run-parts --regex "test*" graphistry-cli/graphistry/bootstrap/ubuntu-cuda9.2
```

Ensure tests pass for `test-10` through `test-40`.

4. Graphistry Container Installation

Load the Graphistry containers into your system's registry:

`docker load -i containers.tar`

5. Start

Launch with `docker-compose up`, and stop with `ctrl-c`. To start as a background daemon, use `docker-compose up -d`.

Congratulations, you have installed Graphistry!

For a demo, try going to `http://MY_SITE/graph/graph.html?dataset=Twitter`, and compare to the public version.

Chapter 3

Configuration

See configure.md for connectors (Splunk, ElasticSearch, . . .), passwords, ontology (colors, icons, sizes), TLS/SSL/HTTPS, backups to disk, and more.

Chapter 4

Maintenance

AWS Marketplace

See AWS Marketplace Administration

OS Restarts

Graphistry automatically restarts in case of errors. In case of manual restart or reboot:

- On reboot, you may need to first run:
- `sudo systemctl start docker`
- If using daemons:
- `docker-compose restart`
- `docker-compose stop` and `docker-compose start`
- Otherwise `docker-compose up`

Upgrading

1. Backup any configuration and data: `.env`, `docker-compose.yml`, `data/*`, `etc/ssl`
2. Stop the Graphistry server if it is running: `docker-compose stop`
3. Load the new containers (e.g., `docker load -i containers.tar`)
4. Edit and reload any config (`docker-compose.yml`, `.env`, `data/*`, `etc/ssl`)
5. Restart Graphistry: `docker-compose up` (or `docker-compose up -d`)

Chapter 5

Testing

- `docker ps` reports no “unhealthy”, “restarting”, or prolonged “starting” services
- Nvidia infrastructure setup correctly
- `nvidia-smi` reports available GPUs

- `docker run --runtime=nvidia nvidia/cuda nvidia-smi` reports available GPUs
- `docker run --rm nvidia/cuda  nvidia-smi` reports available GPUs
- `docker run graphistry/cljs:1.1 npm test` reports success (see air-gapped alternative as well)
- “docker run –rm grph/streamgl-gpu:`cat VERSION`-dev nvidia-smi” reports available GPUs
- Pages load when logged in
- `site.com` shows Graphistry homepage
- `site.com/graph/graph.html?dataset=Facebook` clusters and renders a graph
- `site.com/pivot` loads a list of investigations
- `site.com/pivot/connectors` loads a list of connectors
- ˆ When clicking the `Status` button for each connector, it reports green
- Opening and running an investigation in `site.com/pivot` uploads and shows a graph
- Notebooks
- Running the analyst notebook example generates running visualizations (see logged-in homepage)
- For further information about the Notebook client, see the OSS project PyGraphistry (PyPI).

Chapter 6

Troubleshooting

See further documentation.

Chapter 7

Recommended Deployment Configurations: Client, Server Software, Server Hardware

The recommended non-Enterprise configuration is AWS Marketplace for the server and comes fully configured.

Graphistry Enterprise ships as a Docker container that runs in a variety of Linux + Nvidia GPU environments:

Contents

Overview

- **Client**: Chrome/Firefox from the last 3 years, WebGL enabled, and 100KB/s download ability
- **Server**:

- Minimal: x86 Linux server with 4+ CPU cores, 16+ GB CPU RAM (3GB per concurrent user), and 1+ Nvidia GPUs (P100 onwards) with 4+ GB RAM each (1+ GB per concurrent user)
- Recommended: Ubuntu 16.04, 4+ CPU cores, 64GB+ CPU RAM, Nvidia Pascal or later (Volta, RTX, . . .)
- Docker / CUDA 10 / nvidia-docker-2

Client

A user's environment should support Graphistry if it supports Youtube, and even better, Netflix.

The Graphistry client runs in standard browser configurations:

- **Browser**: Chrome and Firefox from the last 3 years, and users regularly report success with other browsers like Safari.
- **WebGL**: WebGL 1.0 is required. It is 7+ years old, so most client devices, including phones and tablets, support it. Graphistry runs fine on both integrated and discrete graphic cards, with especially large graphs working better on better GPUs.
- **Network**: 100KB+/s download speeds, and we recommend 1MB+/s if often using graphs with 100K+ nodes and edges.
- **Operating System**: All.

Recommended: Chrome from last 2 years on a device from the last 4 years and a 1MB+/s network connection

Server Software: Cloud, OS, Docker, Avoiding Root Users

Graphistry can run both on-premises and in the cloud on Amazon EC2, Google GCP, and Microsoft Azure.

Cloud

Tested AWS Instances:

- P3 ***Recommended for testing and initial workloads***

Tested Azure Instances:

- NV6v2 ***Recommended for testing and initial workloads***
- NC6v2

See the hardware provisioning section to pick the right configuration for you.

OS & Docker

Graphistry runs preconfigured with a point-and-click launch on Amazon Marketplace. Please contact for the latest options for major cloud providers.

Graphistry regularly runs on:

- Ubuntu Xenial 16.04 LTS ***Recommended***
- RedHat RHEL 7.3

Both support nvidia-docker-2:

- Docker
- nvidia-docker-2
- CUDA 10

User: Root vs. Not, Permissions

Installing Docker, Nvidia drivers, and nvidia-docker currently all require root user permissions.

Graphistry can be installed and run as an unprivileged user as long as it have access to nvidia-docker.

Storage

We recommend using backed-up network attached storage for persisting visualizations and investigations. Data volumes are negligible in practice, e.g., < $10/mo on AWS S3.

Server: Hardware Capacity Planning

Graphistry utilization increases with the number of concurrent visualizations and the sizes of their datasets. Most teams will only have a few concurrent users and a few concurrent sessions per user. So, one primary server, and one spillover or dev server, gets a team far.

For teams doing single-purpose multi-year purchases, we generally recommend more GPUs and more memory: As Graphistry adds further scaling features, users will be able to upload more data and burst to more devices.

Network

A Graphistry server must support 1MB+/s per expected concurrent user. A moderately used team server may stream a few hundred GB / month.

- The server should allow http/https access by users and ssh by the administrator.
- TLS certificates can be installed (nginx)
- The Graphistry server may invoke various database connectors: Those systems should enable firewall and credential access that authorizes authenticated remote invocations from the Graphistry server.

GPUs & GPU RAM

The following Nvidia GPUs, Pascal and later, are known to work with Graphistry:

- P100, V100, RTX
- DGX and DGX2

The GPU should provide 1+ GB of memory per concurrent user.

CPU Cores & CPU RAM

CPU cores & CPU RAM should be provisioned in proportion to the number of GPUs and users:

- CPU Cores: We recommend 4-6 x86 CPU cores per GPU
- CPU RAM: We recommend 6 GB base memory and at least 16 GB total memory for a single GPU system. For balanced scaling, 3 GB per concurrent user or 3X the GPU RAM.

Multi-GPU, Multi-Node, and Multi-Tenancy

Graphistry virtualizes a single GPU for shared use by multiple users.

- When Graphistry is on a shared system, it is especially crucial to determine whether the system environment is ready for nvidia-docker-2, or needs potentially disruptive patching updates. Likewise, the CPU, GPU, and network resources assigned to the Graphistry instance (such as via Docker) should not be contended with from sibling applications. Such software is often not as isolatable.

- Multitenancy via multiple GPUs: You can use more GPUs to handle more users and give more performance isolation between users. We recommend separating a few heavy users from many light users, and developers from non-developers.

- Acceleration via multiple GPUs: Graphistry is investigating how to achieve higher speeds via multi-GPU acceleration, but the current benefits are only for multitenancy.

Chapter 8

Graphistry on AWS: Environment Setup Instructions

Graphistry runs on AWS EC2. This document describes initial AWS virtual machine environment setup. From here, proceed to the general Graphistry installation instructions linked below.

The document assumes light familiarity with how to provision a standard CPU virtual machine in AWS.

For AWS Marketplace users, instead see AWS Marketplace Administration

Contents:

1. Pick Linux distribution: Ubuntu 16.04 (Others supported, but not by our nvidia drivers bootstrapper)
2. Configure instance
3. General installation

Subsequent reading: General installation

Chapter 9

1. Pick Linux distribution

Start with one of the following Linux distributions, and configure it using the instructions below under ‘Configure instance’.

Ubuntu 16.04 LTS

- Available on official AWS launch homepage
- Find AMI for region https://cloud-images.ubuntu.com/locator/
- Ex: Amazon AWS us-east-1 xenial 16.04 amd64 hvm-ssd 20180405 ami-6dfe5010
- Follow provisioning instructions from AWS install
- P3.x (Pascal or later): 200 GB, add a name tag, ssh/http/https; use & store an AWS keypair
- Login: ssh -i ... private_key.pem ubuntu@public.dns

RHEL, CentOS temporarily not supported by our bootstrapper while conflicting nvidia-docker<>CUDA changes get fixed in the Linux ecosystem

Chapter 10

2. Configure instance

- Instance: p*
- 200GB+ RAM
- Security groups: ssh, http, https

Chapter 11

3. General installation

Proceed to the instructions for general installation.

Chapter 12

Graphistry in AWS Marketplace

Launching Graphistry in AWS Marketplace? Get started with the walkthrough tutorial and videos!

Advanced administration

The Graphistry marketplace instance is designed for secure web-based use and administration. However, command-line administration can be helpful. This document shares common marketplace tasks. See the main docs for general CLI use.

Contents:

1. **Recommended configuration**
2. **Solve GPU availability errors**
3. **Log in**
4. **Docker**
5. **Install Python packages**
6. **Install native packages**
7. **Marketplace FAQ**

1. Recommended configuration

- Associate your AWS instance with an Elastic IP or a domain
- Setup TLS

2. Solve GPU availability errors

Upon trying to launch, Amazon may fail with an error about no available GPUs for two reasons:

- Lack of GPU availability in the current region. In this case, try another valid GPU type, or launching in another region. For example, Virginia => Oregon. Keeping the GPU close to your users is a good idea to minimize latency.
- Insufficient account quota. In this case, the error should also contain a link to increase your quota. Request `p3.2` (and above), and 1-2 for a primary region and 1-2 for a secondary region.

3. Log in

Log in using the key configured at AWS instance start and your instance's public IP/domain:

```
ssh -i my_key.pem ubuntu@MY_PUBLIC_IP_HERE
```

Many `ssh` clients may require you to first run `chmod 400 my_key.pem` or `chmod 644 my_key.pem` before running the above.

4. Docker

Graphistry leverages `docker-compose` and `nvidia-docker2`.

```
cd ~/graphistry
docker-compose ps
```

=>

```
               Name                             Command                  State
--------------------------------------------------------------------------------------------
graphistry_celerybeat_1              /entrypoint bash /start-ce ...   Up             8080/tc
graphistry_celeryworker_1            /entrypoint bash /start-ce ...   Up             8080/tc
graphistry_forge-etl_1               /tini -- /entrypoints/fast ...   Up (healthy)   8080/tc
graphistry_nexus_1                   /entrypoint /bin/sh -c bas ...   Up             8080/tc
graphistry_nginx_1                   nginx -g daemon off;             Up             0.0.0.0
graphistry_notebook_1                /bin/sh -c graphistry_api_ ...   Up             8080/tc
graphistry_postgres_1                docker-entrypoint.sh postgres    Up             5432/tc
graphistry_redis_1                   docker-entrypoint.sh redis ...   Up             6379/tc
graphistry_streamgl-datasets_1       /tini -- /entrypoints/fast ...   Up (healthy)   8080/tc
graphistry_streamgl-gpu_1            /tini -- /entrypoints/fast ...   Up (healthy)   8080/tc
graphistry_streamgl-sessions_1       /tini -- /entrypoints/fast ...   Up (healthy)   8080/tc
```

```
graphistry_streamgl-svg-snapshot_1   /tini -- /entrypoints/fast ...   Up (healthy)   8080/tc
graphistry_streamgl-vgraph-etl_1     /tini -- /entrypoints/fast ...   Up (healthy)   8080/tc
graphistry_streamgl-viz_1            /tini -- /entrypoints/stre ...   Up             8080/tc
```

5. Install Python packages

If you see `wheel` errors, you may need to run `pip3 install wheel` and restart your Jupyter kernel.

6. Install native packages

By default, Jupyter users do not have `sudo`, restricting them to user-level installation like `pip`. For system-level actions, such as for installing `golang` and other tools, you can create interactive `root` user sessions in the Jupyter Docker container:

Admin:

Note that `sudo` is unnecessary:

```
ubuntu@ip-172-31-0-38:~/graphistry$ docker exec -it -u root graphistry_notebook_1 bash
root@d4afa8b7ced5:/home/graphistry# apt update
root@d4afa8b7ced5:/home/graphistry# apt install golang
```

User:

```
ubuntu@ip-172-31-0-38:~/graphistry$ docker exec -it  graphistry_notebook_1 bash
graphistry@d4afa8b7ced5:~$ go version
```

=>

```
go version go1.10.4 linux/amd64
```

7. Marketplace FAQ

No site loads or there is an Nginx 404 error

Wait a few minutes for the system to finish starting. If the problem persists for more than 5-10min, log in, run `docker ps`, and for each failing service, restart it. If problems persist further, please report the results of `docker logs <service>` to the Graphistry support team and we will help out.

I lost my admin account

See the `reset` command in the main README (requires logging in)

I want to log into the server

See section `log in`

See general installation for further information.

Chapter 13

Graphistry on Azure: Environment Setup Instructions

Graphistry runs on Azure. This document describes initial Azure virtual machine environment setup. From here, proceed to the general Graphistry installation instructions linked below.

The document assumes light familiarity with how to provision a standard CPU virtual machine in Azure.

Contents:

- Prerequisites: Azure GPU Quota
  - Testing if you already have GPU Quota
  - Requesting Azure for GPU Quota

1. Start a new GPU virtual machine
2. Proceed to general Graphistry installation

Subsequent reading: General installation

Prerequisites: Azure GPU Quota

You may need to make quota requests to add GPUs to each of your intended locations:

- **Minimal GPU type**: NC6 (hdd) in your region
- **Maximal GPU type**: N-Series, see general documentation for sizing considerations

Testing if you already have GPU quota

Go through the **Start a new GPU virtual machine**, then tear it down if successful

Requesting Azure for GPU Quota

For each location in which you want to run Graphistry:

1. Start help ticket: `? (Help)` -> `Help + support` -> `New support request`
2. Fill out ticket
3. **Basics**: `Quota` -> `<Your Subscription>` -> `Compute (cores/vCPUs)` -> `Next`
4. **Problem**: Specify location/SKU, e.g., `West US 2` or `East US` for `NC Series`
5. **Contact Information**: Fill out and submit

Expect 1-3 days based on your requested `Severity` rating and who Azure assigns to your ticket

1. Start a new GPU virtual machine

See general installation instructions for currently supported Linux versions (subject to above Azure restrictions and general support restrictions.)

1. **Virtual machines** -> `Create virtual machine`
2. **Ubuntu 16.04 LTS** Please let us know if another OS is required
3. **Basics**: As desired; make sure can login, such as by SSH public key; needs to be a region with GPU quota
4. **Size**: GPU of all disk types, e.g., NC6 (hdd) is cheapest for development
5. **Settings**: Open ports for administration (SSH) and usage (HTTP, HTTPS)
6. **Summary**: Should say "`Validation passed`" at the top -> visually audit settings + hit `Create`

2. Confirm proper instance

1. Test login; see SSH command at `Overview` -> `Connect` -> `Login using VM Account`
2. Check to make sure GPU is attached:

```
$ lspci -vnn | grep VGA -A 12
0000:00:08.0 VGA compatible controller [0300]: Microsoft Corporation Hyper-V virtual VGA [14
    Flags: bus master, fast devsel, latency 0, IRQ 11
    Memory at f8000000 (32-bit, non-prefetchable) [size=64M]
    [virtual] Expansion ROM at 000c0000 [disabled] [size=128K]
    Kernel driver in use: hyperv_fb
    Kernel modules: hyperv_fb

5dc5:00:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev a
    Subsystem: NVIDIA Corporation GK210GL [Tesla K80] [10de:106c]
    Flags: bus master, fast devsel, latency 0, IRQ 24, NUMA node 0
    Memory at 21000000 (32-bit, non-prefetchable) [size=16M]
    Memory at 1000000000 (64-bit, prefetchable) [size=16G]
    Memory at 1400000000 (64-bit, prefetchable) [size=32M]
```

3. Proceed to general Graphistry installation

Login to your instance (see **Test login** above) and use the instructions for general installation.

For steps involving an IP address, see needed IP value at Azure console in `Overview` -> `Public IP address`

Chapter 14

Graphistry Data Bridge for Proxying

Graphistry now supports bridged connectors, which eases tasks like crossing from a cloud server to on-premise databases. It is designed to work with enterprise firewall policies such as HTTP-only and outgoing-only.

Prerequisites

- Running Graphistry server, with admin access
- Proxy installer, access to proxy server, `docker`, and `docker-compose`

Design

Graphistry server

- Bridged connectors: Built into standard Graphistry servers
- Individual database connectors can be configured to use a proxy instead of a direct connection

Graphistry proxy

- Separate install:
- Proxies establish persistent outgoing http/https connections with your Graphistry server. This enables the server to quickly push queries to your proxy.

Keys

- For each connector, generate unguessable UUIDs for the server and client - this enables either side to trust and revoke access

Example: Splunk

Generate a key

Ex: Uisng your Graphistry server:

```
docker exec -it ec2-user_pivot_1 node_modules/uuid/bin/uuid v4 =>
<my_key_1>
```

Server

In `.env`, setup Splunk connector and set it to bridge via a Graphistry proxy:

```
SPLUNK_HOST=splunk.example.com
SPLUNK_WEB_PORT=3000
SPLUNK_USER=my_user
SPLUNK_KEY=my_pwd
### SPLUNK PROXY ###
SPLUNK_USE_PROXY=true
SPLUNK_SERVER_KEY=my_key_1
SPLUNK_PROXY_KEY=my_key_2
```

Proxy

Sample edits to `docker-compose.yml`:

```
services:
  agents:
    ...
    environment:
      #REQUIRED: Fill in with your server
      - GRAPHISTRY_HOST=http://graphistry.mysite.com
      #LIKELY: Fill in for connectors this proxy provides
      - SPLUNK_USE_PROXY=true
      - SPLUNK_PROXY_KEY=my_key_2
      - SPLUNK_SERVER_KEY=my_key_1
      #STANDARD
      - GRAPHISTRY_LOG_LEVEL=DEBUG
```

Install and launch proxy

1. Install

   ```
   tar -xvvf proxy.tar.gz
   cd proxy
   docker load -i proxy.tar
   ```

2. Configure

Edit `.env` and `docker-compose.yml`. See above example.

3. Launch

```
docker-compose up -d
```

Chapter 15

Browser Configuration & Debugging

Graphistry is optimized from Chrome (Safari, new IE) and supports Firefox

It runs on mobile and tablets, and is subject to the device memory

Symptom: Missing nodes/edges

- Check that WebGL 1.0 is enabled
- Ensure that the window size is not too big
- Check the filter and exclude panels are not hiding data

Symptom: The browser crashes

- Try a smaller graph
- Check that WebGL is using hardware acceleration, not software emulation
- Give the browser more JS and WebGL memory
- In OS X: `open /Applications/Google\ Chrome.app --args --js-flags="--max_old_space_size=8`
- Use a client device with a dedicated GPU and more GPU memory

Chapter 16

Configure Investigations

Many Graphistry investigation configurations can be set through environment variables (your `.env`), in your `.pivot-db/config/config.json`, or in the admin panel.

These control aspects including: * Connector auth and defaults: Splunk, Neo4j, . . . * Layouts * Ontology: column->type mapping, colors, icons, sizes, . . . * Prepopulated investigation steps

After editing, restart your server, or at least `pivot`.

For broader configuration information, see the main configuration docs.

Chapter 17

Example

Set log level to debug:

Via `.env`:

```
GRAPHISTRY_LOG_LEVEL=DEBUG
```

Via `config.json`:

```
{
  "log": {
    "level": "DEBUG"
  }
}
```

After setting these, restart your server.

Chapter 18

Schema

```
{
    "env": {
        "doc": "The applicaton environment.",
        "format": {
            "0": "production",
            "1": "development",
            "2": "test"
        },
        "default": "development",
        "env": "NODE_ENV"
    },
    "host": {
        "doc": "Pivot host name/IP",
        "format": "ipaddress",
        "default": "0.0.0.0",
        "env": "PIVOT_HOST_IP"
    },
    "port": {
        "doc": "Pivot port number",
        "format": "port",
        "default": 8080,
        "arg": "port",
        "env": "PORT"
    },
    "layouts": {
        "network": {
            "ipInternalAcceptList": {
                "doc": "Array of strings and JavaScript regexes for IPs considered internal
                "format": "array",
                "arg": "internal-ips",
                "env": "PIVOT_INTERNAL_IP_ACCEPTLIST"
            }
        },
        "parallelCoordinates": {
            "orders": {
                "doc": "JSON dictionary naming axis column name orders. Defaults to key 'def
                "format": "object",
                "default": {},
                "arg": "parallel-coords-axes",
                "env": "GRAPHISTRY_PARALLEL_COORDS_AXES"
            }
        }
    },
    "authentication": {
        "passwordHash": {
            "doc": "Bcrypt hash of the password required to access this service, or unset/en
            "format": "string",
            "default": "",
            "arg": "password-hash",
            "env": "PIVOT_PASSWORD_HASH",
            "sensitive": true
        },
        "username": {
            "doc": "The username used to access this service",
            "format": "string",
            "default": "admin",
            "arg": "username",
            "env": "PIVOT_USERNAME"
        }
    },
    "features": {
        "axes": {
            "format": "boolean",
            "default": true
        }
    },
    "systemTemplates": {
        "pivots": {
            "doc": "JSON list of pivots:\n                [{template, name, id, tags: [Strin
            "format": "array",
            "default": {},
            "arg": "pivots",
            "env": "GRAPHISTRY_PIVOTS"
        }
    },
    "ontology": {
        "icons": {
            "doc": "JSON dictionary from entity type to icon name:\n                { \"myTy
            "format": "object",
            "default": {},
            "arg": "icons",
            "env": "GRAPHISTRY_ICONS"
        },
        "colors": {
            "doc": "JSON dictionary from entity type to color hex code:\n                { \
            "format": "object",
            "default": {},
            "arg": "colors",
            "env": "GRAPHISTRY_COLORS"
        },
        "sizes": {
            "doc": "JSON dictionary from entity type to size integers (1-1000), with Graphis
            "format": "object",
            "default": {},
            "arg": "sizes",
            "env": "GRAPHISTRY_SIZES"
        },
        "products": {
            "doc": "JSON dictionary of per-product encodings:\n              {\n
            "format": "object",
            "default": {},
            "arg": "products",
            "env": "GRAPHISTRY_PRODUCTS"
        }
    },
    "pivotApp": {
        "mountPoint": {
            "doc": "Pivot mount point",
            "format": "string",
            "default": "/pivot",
            "arg": "pivot-mount-point"
        },
        "cachePoint": {
            "doc": "Nginx caching point",
            "format": "string",
            "default": "/cached",
            "arg": "pivot-cache-point"
        },
        "dataDir": {
            "doc": "Directory to store investigation files",
            "format": "string",
            "default": "data",
            "arg": "pivot-data-dir"
        }
    },
    "log": {
        "level": {
            "doc": "Log levels - ['TRACE', 'DEBUG', 'INFO', 'WARN', 'ERROR', 'FATAL']",
            "format": {
                "0": "TRACE",
                "1": "DEBUG",
                "2": "INFO",
                "3": "WARN",
                "4": "ERROR",
                "5": "FATAL"
            },
            "default": "INFO",
            "arg": "log-level",
            "env": "GRAPHISTRY_LOG_LEVEL"
        },
        "file": {
            "doc": "Log so a file intead of standard out",
            "format": "string",
            "arg": "log-file",
            "env": "LOG_FILE"
        },
        "logSource": {
            "doc": "Logs line numbers with debug statements. Bad for Perf.",
            "format": "boolean",
            "default": false,
            "arg": "log-source",
            "env": "LOG_SOURCE"
        }
    },
    "graphistry": {
        "key": {
            "doc": "Graphistry's api key",
            "format": "string",
            "arg": "graphistry-key",
            "env": "GRAPHISTRY_KEY",
            "sensitive": true
        },
        "host": {
            "doc": "The location of Graphistry's Server",
            "format": "string",
            "default": "http://graphistry",
            "arg": "graphistry-host",
            "env": "GRAPHISTRY_HOST"
        }
    },
    "pivots": {
        "show": {
            "doc": "Pivots to show; undefined means all. See load sequence output for availa
            "format": "array",
            "arg": "pivots-show",
            "env": "PIVOTS_SHOW"
        },
        "hide": {
            "doc": "Pivots to hide; undefined means none. See load sequence output for avail
            "format": "array",
            "arg": "pivots-hide",
            "env": "PIVOTS_HIDE"
        }
    },
    "neo4j": {
        "bolt": {
            "doc": "Neo4j BOLT endpoint, e.g., bolt://...:24786",
            "format": "string",
            "arg": "neo4j-bolt",
            "env": "NEO4J_BOLT"
        },
        "user": {
            "doc": "Neo4j user name",
            "format": "string",
            "arg": "neo4j-user",
            "env": "NEO4J_USER"
        },
        "password": {
            "doc": "Neo4j password",
            "format": "string",
            "arg": "neo4j-password",
            "env": "NEO4J_PASSWORD",
            "sensitive": true
        }
    },
    "elasticsearch": {
        "host": {
            "doc": "The hostname of the Elasticsearch Server",
            "format": "string",
            "arg": "es-host",
            "env": "ES_HOST"
        },
        "port": {
            "doc": "Elasticsearch port",
            "format": "number",
            "default": 9200,
            "arg": "es-port",
            "env": "ES_PORT"
        },
        "version": {
            "doc": "Elasticsearch version as major.minor (6.2, 5.6, ...), autodetects by def
            "format": "string",
            "arg": "es-version",
            "env": "ES_VERSION"
        },
        "protocol": {
            "doc": "HTTP or HTTPS",
            "format": "string",
            "default": "http",
            "arg": "es-protocol",
            "env": "ES_PROTOCOL"
        },
        "auth": {
            "doc": "HTTP credentials -- user:password, or undefined",
            "format": "string",
            "arg": "es-auth",
            "env": "ES_AUTH"
        }
    },
    "vt": {
        "host": {
            "doc": "The VT host, you usually want to leave this alone",
            "format": "string",
            "default": "https://www.virustotal.com"
        },
        "fileReport": {
            "doc": "The file report path, you usually want to leave this alone",
            "format": "string",
            "default": "/vtapi/v2/file/report"
        },
        "key": {
            "doc": "The VT key, you might want one",
            "format": "string",
            "sensitive": false
        }
    },
    "splunk": {
        "key": {
            "doc": "Splunk password",
            "format": "string",
            "default": "admin",
            "arg": "splunk-key",
            "env": "SPLUNK_KEY",
            "sensitive": true
        },
        "user": {
            "doc": "Splunk user name",
            "format": "string",
            "default": "admin",
            "arg": "splunk-user",
            "env": "SPLUNK_USER"
        },
        "host": {
            "doc": "The hostname of the Splunk Server (splunk.example.com)",
            "format": "string",
            "arg": "splunk-host",
            "env": "SPLUNK_HOST"
        },
        "port": {
            "doc": "Splunk API port",
            "format": "number",
            "default": 8089,
            "arg": "splunk-port",
            "env": "SPLUNK_PORT"
        },
        "uiPort": {
            "doc": "Splunk web UI port",
            "format": "number",
            "default": 443,
            "arg": "splunk-web-port",
            "env": "SPLUNK_WEB_PORT"
        },
        "scheme": {
            "doc": "Splunk protocol",
            "format": {
                "0": "http",
                "1": "https"
            },
            "default": "https",
            "arg": "splunk-scheme",
            "env": "SPLUNK_SCHEME"
        },
        "suffix": {
            "doc": "Splunk url suffix, e.g., en-US in mysplunk.com/en-US/app/search",
            "format": "string",
            "default": "/en-US",
            "arg": "suffix",
            "env": "SPLUNK_SUFFIX"
        },
        "jobCacheTimeout": {
            "doc": "Time (in seconds) during which Splunk caches the query results. Set to -
            "format": "number",
            "default": 14400,
            "arg": "splunk-cache-timeout",
            "env": "SPLUNK_CACHE_TIMEOUT"
        },
        "searchMaxTime": {
            "doc": "Maximum time (in seconds) allowed for executing a Splunk search query.",
            "format": "number",
            "default": 20,
            "arg": "splunk-search-max-time",
            "env": "SPLUNK_SEARCH_MAX_TIME"
        }
    }
}
```

Chapter 19

Configuring Graphistry

Administrators can specify passwords, TLS/SSL, persist data across sessions, connect to databases, specify ontologies, and more.

For a list of many investigation-oriented options, see their settings reference page.

Top configuration places: .env, .pivot-db/config/config.json

- Graphistry is primarily configured through a `.env` file
- Richer ontology configuration is optionally via `.pivot-db/config/config.json`. Many relevant options are detailed in a reference page.

Between edits, restart one or all Graphistry services: `docker-compose stop` and `docker-compose up -d`

Further configuration: docker-compose.yml, Caddyfile, and etc/ssl/*

- More advanced administrators may edit `docker-compose.yml` . Maintenance is easier if you never edit it.
- TLS is via editing `Caddyfile`(docs, or being phased out, Nginx config (`etc/ssl/*`)

Backup your configuration

Graphistry tarballs contain default `.env` and `.pivot-db/config/config.json`, so make sure you put them in safe places and back them up.

If you configure `TLS`, backup `Caddyfile` or `etc/ssl`.

If you edit `docker-compose.yml` (not encouraged), back that up too.

Backup your data

See `/home/ubuntu/graphistry/data` (default in `docker-compose.yml`), `.pivot-db/investigations`, and `.pivot-db/pivots`

Connectors

Uncomment and edit lines of `.env` corresponding to your connector and restart Graphistry:

```
ES_HOST...
SPLUNK...
```

Ontology

See settings reference page for full options.

Edit `.pivot-db/config/config.json` via the below and restart Graphistry:

- Icons: Use Font Awesome 4 names (https://fontawesome.com/v4.7.0/icons/)
- Colors: Use hex codes (`#vvvvvv`). To find hex values for different colors, you can use Graphistry's in-tool background color picker.

```
{
    "ontology": {
        "products": {
            "myOntologyName": {
                "colTypes": {
                    "src_ip": "ip",
                    "dest_ip": "ip",
                    "myEventColumnName": "myTypeTag"
                }
            }
        },
        "icons": {
            "ip": "laptop",
            "myTypeTag": "fighter-jet"
        },
        "sizes": {
            "ip": 800,
            "myTypeTag": 100
        },
        "colors": {
            "ip": "#FF0000",
            "myTypeTag": "#000000"
        }
    }
}
```

TLS: Caddyfile and Nginx Config

To simplify credentials deployment, Graphistry is moving from Nginx to Caddy:

Caddyfile

For automatic TLS (Let's Encrypt) and manual certs, see official docs

Nginx

There are two helper ssl configs provided for you in the `./etc/nginx` folder.

ssl.self-provided.conf

```
listen 443 ssl;
# certs sent to the client in SERVER HELLO are concatenated in ssl_certificate
# Includes the website cert, and the CA intermediate cert, in that order
ssl_certificate            /etc/ssl/ssl.crt;

# Unencrypted key file
ssl_certificate_key        /etc/ssl/ssl.key;
```

Notice the location and file names of the SSL keys and certs. Also the SSL include in the supplied `graphistry.conf`.

graphistry.conf

```
...
```

```
    server_name                 _;

    proxy_http_version          1.1;
    client_max_body_size        256M;

    import /etc/nginx/graphistry/ssl.conf

    proxy_set_header            Host              $http_host;
    proxy_set_header            X-Real-IP         $remote_addr;
    proxy_set_header            X-Forwarded-For   $proxy_add_x_forwarded_for;
    proxy_set_header            X-Forwarded-Proto $scheme;

    # Support proxying WebSocket connections
    proxy_set_header            Upgrade           $http_upgrade;
    proxy_set_header            Connection        $connection_upgrade;

    # Block Slack's link preview generator bot, so that posting a viz link into Slack doesn'
    # overwhelm the server. We should have a more robust system for stopping all bots, thoug
    if ($http_user_agent ~* Slack) {
        return 403;
    }

...
```

If you uncomment the nginx volume mounts in the `docker-compose.yml` and supply SSL key and certs, SSL will start right up for you.

docker-compose.yml

```
  nginx:
    ports:
      - 80:80
      - 443:443
    links:
      - pivot
      - central
    # volumes:
    #   - ./etc/nginx/nginx.conf:/etc/nginx/nginx.conf
    #   - ./etc/nginx/graphistry.conf:/etc/nginx/conf.d/graphistry.conf
    #   - ./etc/nginx/ssl.self-provided.conf:/etc/nginx/graphistry/ssl.conf
    #   - ./etc/ssl:/etc/ssl
```

There is an alternate SSL conf you can use if yo uare not using a self signed cert. `./etc/nginx/ssl.conf`.

We have a helper tool for generating self signed ssl certs that you can use by running:

```
bash scripts/generate-ssl-certs.sh
```

Chapter 20

Debugging Container Networking

The following tests may help pinpoint loading failures.

Prerequisites

Check the main tests (https://github.com/graphistry/graphistry-cli)

- All containers are running
- Healthchecks passes

Mongo container

A. Host is running Mongo

Note: Database, collection initializated by `launch` (e.g., during `init`) and does not persist between runs.

```
docker exec monolith-network-mongo /bin/bash -c "echo 'db.stats().ok' | mongo localhost/clus
```

=>

```
1
```

B. Mongo has registered workers

Note: Populated by `monolith-network-viz` on node process start

```
docker exec monolith-network-mongo /bin/bash -c "echo 'db.node_monitor.find()' | mongo local
```

=> 10+ workers

```
{ "_id" : ObjectId("5b4d7049f160a28b5001a6bf"), "ip" : "localhost", "pid" : 9867, "port" : 1
{ "_id" : ObjectId("5b4d727ff160a28b5001a6c3"), "ip" : "localhost", "pid" : 13325, "port" :
{ "_id" : ObjectId("5b4d729af160a28b5001a6c4"), "ip" : "localhost", "pid" : 13392, "port" :
{ "_id" : ObjectId("5b4d72e0f160a28b5001a6ca"), "ip" : "localhost", "pid" : 13966, "port" :
{ "_id" : ObjectId("5b4d7306f160a28b5001a6cc"), "ip" : "localhost", "pid" : 14156, "port" :
{ "_id" : ObjectId("5b4d75fff160a28b5001a6d0"), "ip" : "localhost", "pid" : 17872, "port" :
...
```

Browser

A. Can access site:

Browse to

```
curl http://MY_GRAPHISTRY_SERVER.com/central/healthcheck
```

=>

```
{"success":true,"lookup_id":"<NUMBER>","uptime_ms":<NUMBER>,"interval_ms":<NUMBER>}
```

B. Browser has web sockets enabled

Passes test at https://www.websocket.org/echo.html

C. Can follow central redirect:

Open browser developer network analysis panel and visit

```
http://MY_GRAPHISTRY_SERVER.com/graph/graph.html?dataset=Twitter
```

=>

```
302 on `/graph/graph.html?dataset=Twitter
200 on `/graph.graph.html?dataset=Twitter&workbook=<HASH>`
Page UI loads (`vendor.<HASH>.css`, ...)
Socket connects (`/worker/<NUMBER>/socket.io/?dataset=Twitter&...`)
Dataset positions stream in (`/worker/<NUMBER>/vbo?id=<HASH>&buffer=curPoints`)
```

This call sequence stress a lot of the pipeline.

NGINX

Note Assumes underlying containers are fulfilling these requests (see other tests)

A. Can server central routes

```
curl -s -I localhost/central/healthcheck | grep HTTP
```

=>

```
HTTP/1.1 200 OK
```

B. Can receive central redirect:

```
curl -s -I localhost/graph/graph.html?dataset=Twitter | grep "HTTP\|Location"
```

=>

```
HTTP/1.1 302 Found
Location: /graph/graph.html?dataset=Twitter&workbook=<HASH>
```

and

```
curl -s -I localhost/graph/graph.html?dataset=Twitter  |  grep "HTTP\|Location"
```

=>

```
HTTP/1.1 302 Found
Location: /graph/graph.html?dataset=Twitter&workbook=<HASH>
```

C. Can serve worker routes

```
curl -s -I localhost/worker/10000/healthcheck | grep HTTP
```

=>

```
HTTP/1.1 200 OK
```

Viz container

A. Container has a running central server

```
docker exec monolith-network-viz curl -s -I localhost:3000/central/healthcheck | grep HTTP
```

=>

```
HTTP/1.1 200 OK
```

and

```
docker exec monolith-network-viz curl -s -I localhost:3000/graph/graph.html?dataset=Twitter
```

=>

```
HTTP/1.1 302 Found
Location: /graph/graph.html?dataset=Twitter&workbook=<HASH>
```

C. Can communicate with Mongo

First find mongo configuration for MONGO_USERNAME and MONGO_PASSWORD:
`docker exec monolith-network-viz cat central-cloud-options.json` or
`docker exec  monolith-network-viz ps -eafww | grep central`

Plug those into `<MONGO_USERNAME>` and `<MONGO_PASSWORD>` below:

```
docker exec -w /var/graphistry/packages/central monolith-network-viz node -e "x = require('r
```

=>

```
ok [ { _id: <HASH>,
    ip: 'localhost',
    pid: <NUMBER>,
    port: <NUMBER>,
    active: true,
    updated: <TIME> },
  { _id: <HASH>,
    ip: 'localhost',
    pid: <NUMBER>,
    port: <NUMBER>,
    active: true,
    updated: <TIME> },
...
```

D. Has running workers

```
docker exec monolith-network-viz curl -s -I localhost:10000/healthcheck | grep HTTP
```

=>

```
HTTP/1.1 200 OK
```

Chapter 21

Graphistry System Debugging FAQ

Issues sometimes occur during server start, especially in on-premises scenarios with environment configuration drift.

List of Issues

1. Started before initialization completed
2. GPU driver misconfiguration
3. Wrong or mismatched containers installed

1. Issue: Started before initialization completed

Primary symptom

Visualization page never returns or Nginx “504 Gateway Time-out” due to services still initializing." Potentially also “502”.

Correlated symptoms

- GPU tests pass

- Often with first-ever container launch
- Likely within 60s of launch
- Can happen even after static homepage loads

- In `docker-compose up` logs (or `docker logs ubuntu_central_1`):
- "Error: Server at maximum capacity…
- "Error: Too many users…
- "Error while assigning…

Solution

- Try stopping and starting the containers
- Wait for 1-2min after start and try again
- Viz container should report a bunch of `INFO success: viz-worker-10006 entered RUNNING state, process has stayed up for > than 1 seconds (startsecs)`
- Mongo container should report a bunch of `I ACCESS  [conn66] Successfully authenticated as principal graphistry on cluster`

2. Issue: GPU driver misconfiguration

Primary symptoms

- Visualization page never returns or Nginx "504 Gateway Time-out" due to services failing to initialize GPU context. Potentially also "502".
- Visualization loads and positions appear, but never starts clustering, and browser console reports a web socket disconnect

Correlated symptoms

- `node` processes in `ubuntu_viz_1` container fail to run for more than 30s (check durations through `docker exec -it ubuntu_viz_1 ps "-aux"`)
- Upon manually starting a worker in `ubuntu_viz_1`, error message having to do with GPUs (Nvidia, OpenCL, drivers, context, …)
- `docker exec -it ubuntu_viz_1 bash -c "VIZ_LISTEN_PORT=7000 node /opt/graphistry/apps/core/viz/index.js"`
- GPU tests fail
- host
  - `nvidia-smi`
  - Failure: host has no GPU drivers
  - Optional: See https://www.npmjs.com/package/@graphistry/cljs

  - *note*: Requires CL installed in host, which production use of Graphistry does not require
- container

  - ./graphistry-cli/graphistry/bootstrap/ubuntu-cuda9.2/test-20-docker.sh
  - ./graphistry-cli/graphistry/bootstrap/ubuntu-cuda9.2/test-30-CUDA.sh
  - ./graphistry-cli/graphistry/bootstrap/ubuntu-cuda9.2/test-40-nvidia-docker.sh
  - nvidia-docker run –rm nvidia/cuda nvidia-smi
  - nvidia-docker exec -it ubuntu_viz_1 nvidia-smi
  - If `run --rm nvidia/cuda` succeeds but `exec` fails, you likely need to update `/etc/docker/daemon.json` to add `nvidia-container-runtime`, and `sudo service docker restart`, and potentially clean stale images to make sure they use the right runtime
  - See https://www.npmjs.com/package/@graphistry/cljs
  - In container `ubuntu_viz_1`, create & run `/opt/graphistry/apps/lib/cljs/test/cl`
    `node test-nvidia.js`:
    ```
    const cl = require('node-opencl');
    const { argv } = require('../util');
    const { CLPlatform, CLDeviceTypes } = require('../../');
    CLPlatform.devices('gpu')[0].isNvidiaDevice === true
    ```

Solution

- Based on where the issue is according to the above tests, fix that installation layer
- If problems persist, reimaging the full box or switching to a cloud instance may prevent heartache

3. Issue: Wrong or mismatched containers installed

Primary symptom

Especially when upgrading, only some images may have updated. You can delete all of them and start from scratch.

Correlated symptoms

- `docker images` or `docker ps` shows surprising versions

Solution

Delete graphistry images and reinstall * Identify installed images: `docker images | grep graphistry` and `docker images | grep nvidia` * Remove: `docker rmi -f graphistry/nginx-proxy graphistry/graphistry-central ...` * Reload: `docker load -i containers.tar`

Chapter 22

Analyzing Graphistry visual session debug logs

Sometimes visualizations fail to load. This document describes how to inspect the backend logs for loading a visualization and how that may narrow down failures to specific services. For example, if a firewall is blocking file access, the data loader may fail.

It covers the core visualization service. It does not cover the graph upload service nor the investigation template environment.

Prerequisites

- Graphistry starts (seeing `docker ps` section of your install guide) with no restart loops
- Graphistry documentation loads: going to `mygraphistry.com` shows a page similar to `http://labs.graphistry.com/`.
- Logged into system terminal for a Graphistry server

Setup

1. Enable debug logs

In folder `~/`, modify `(httpd|viz-app|pivot-app)-config.json` to turn on debug logs:

```
...
    "log": {
        "level": "debug"
    }
...
```

2. Restart Graphistry (`docker restart <containerid>`)

3. Ensure all workers reported in and are ready:

```
docker exec monolith-network-mongo mongo localhost/cluster --eval
"printjson(db.node_monitor.find({}).toArray())"
```

Should report 32 workers that look like:

```
{
        "_id" : ObjectId("5b5022ab689859b490c6bae3"),
        "ip" : "localhost",
        "pid" : 25,
        "port" : 10001,
        "active" : false,
        "updated" : ISODate("2018-07-20T00:13:38.957Z")
}
```

4. Watch `nginx`, `central`, and `worker` logs:

- `tail -f deploy/nginx/*.log`
- `tail -f deploy/graphistry-json/central.log`
- `tail -f deploy/graphistry-json/viz-worker*.log | grep -iv healthcheck`

Clear screen before starting the test session.

5. Start test session:

Navigate browser to `http://www.yourgraphistry.com/graph/graph.html?dataset=Facebook`

Nginx logs

Nginx in debug mode should log the following sequence of `GET` and `POST` requests. An error or early stop hints at which service is failing. The pipeline is roughly: create a session's workbook, redirect the user to it, starts a GPU service session, loads the static UI, connect a browser's socket to the GPU session, and then starts streaming visual data to the browser.

1. `GET /graph/graph.html?dataset=Facebook`
2. `GET /graph/graph.html?dataset=Facebook&workbook=<SOME_FRAGMENT_STRING>`
3. `GET /worker/<WORKER_NUMBER>/socket.io/?dataset=Facebook&workbook=<SOME_FRAGMENT_STRING>`
4. `GET /worker/<WORKER_NUMBER>/graph/img/logo_white_horiz.png`
5. `5 x GET/POST /worker<WORKER_NUMBER>/socket.io/?dataset=Facebook&workbook=<SOME_FRAGMENT`
6. `GET  /worker/<WORKER_NUMBER>/vbo?...`

Central logs

Central in debug mode should log the successful process of identifying a free worker and redirecting to it. It hints at problems around steps 1 & 2 of the Nginx sequence.

To increase legibility, you can also pipe the JSON logs through a pretty printer like Bunyan.

```
{"name":"graphistry","metadata":{"userInfo":{}},"hostname":"cbf3628eef58","pid":32,"module":
{"name":"graphistry","metadata":{"userInfo":{}},"hostname":"cbf3628eef58","pid":32,"module":
...
{"_id":"5b517fb16e07e97d5d93bf40","ip":"localhost","pid":216,"port":10027,"active":false,"up
{"name":"graphistry","metadata":{"userInfo":{}},"hostname":"cbf3628eef58","pid":32,"module":
...
{"name":"graphistry","metadata":{"userInfo":{}},"hostname":"cbf3628eef58","pid":32,"module":
{"name":"graphistry","metadata":{"userInfo":{}},"hostname":"cbf3628eef58","pid":32,"module":
```

Worker Logs

GPU web session workers in debug mode will report they are climed,

Session handshakes

```
{...,"msg":"HTTP request received by Express.js { originalUrl: '/graph/graph.html?dataset=Fa
{...,"active":true,"msg":"Reporting worker is active.","time":"2018-07-20T06:56:04.336Z","v'

{...,"module":"serv...,"req":{"method":"GET","url":"/graph/graph.html?dataset=Facebook&workb

{...,"req":{"method":"GET","url":"/graph/graph.html?dataset=Facebook&workbook=4425d4d6a7b26f
```

Start hydrating session workbook, GPU configuration

```
{...,"err":{"message":"ENOENT: no such file or directory, stat '/tmp/graphistry/workbook_cac
{...,"err":{"message":"Missing credentials in config","name":"Error","stack":"Error: Missing

{...,"layoutAlgorithms":[{"params":{"tau":{"type":"discrete","displayName":"Precision vs. Sp
{...,"msg":"Attempted to send falcor update, but no socket connected yet.","time":"2018-07-2
```

Load data into backend

```
{"name":"Facebook","metadata":{....,"type":"default","scene":"default","mapper":"default","c
```

```
{...,"msg":"Cannot fetch headers from S3, falling back on cache","time":"2018-07-20T06:56:05
{...,"msg":"Found up-to-date file in cache Facebook","time":"2018-07-20T06:56:05.835Z","v":0
{...,"msg":"Attempted to send falcor update, but no socket connected yet.","time":"2018-07-2
{...,"msg":"Decoding VectorGraph (version: 0, name: , nodes: 4039, edges: 88234)","time":"20
{...,"msg":"Attempted to send falcor update, but no socket connected yet.","time":"2018-07-2
...
{...,"attributes":["label","community_louvain","degree","indegree","outdegree","community_sp
{...,"msg":"Attempted to send falcor update, but no socket connected yet.","time":"2018-07-2
{...,"msg":"Skipping unmapped attribute label","time":"2018-07-20T06:56:05.955Z","v":0}
```

Load data into GPU

```
{...,"msg":"Attempted to send falcor update, but no socket connected yet.","time":"2018-07-2
{...,"msg":"Number of points in simulation: 4039","time":"2018-07-20T06:56:05.958Z","v":0}
{...,"msg":"Creating buffer curPoints, size 32312","time":"2018-07-20T06:56:05.959Z","v":0}
{...,"msg":"Creating buffer nextPoints, size 32312","time":"2018-07-20T06:56:05.959Z","v":0}
{...,"msg":"Attempted to send falcor update, but no socket connected yet.","time":"2018-07-2
{...,"msg":"Number of edges: 88234","time":"2018-07-20T06:56:05.973Z","v":0}
```

```
{...,"msg":"Dataset    nodes:4039  edges:176468  splits:%d","time":"2018-07-20T06:56:06.288Z
{...,"msg":"Number of midpoints:  0","time":"2018-07-20T06:56:06.288Z","v":0}
{...,"msg":"Number of edges in simulation: 88234","time":"2018-07-20T06:56:06.289Z","v":0}
{...,"msg":"Creating buffer degrees, size 16156","time":"2018-07-20T06:56:06.289Z","v":0}
...
{...,"memFlags":1,"map":[1],"msg":"Flags set","time":"2018-07-20T06:56:06.297Z","v":0}
{...,"msg":"Attempted to send falcor update, but no socket connected yet.","time":"2018-07-2
```

```
{...,"msg":"Updating simulation settings { simControls: { ForceAtlas2Barnes: { tau: 0 } } }"
```

Run default backend data pipeline

```
{...,"msg":"Starting Filtering Data In-Place by DataframeMask","time":"2018-07-20T06:56:06.3
```

Connect to browser socket (post-UI-load)

```
{...,"msg":"Socket connected before timeout","time":"2018-07-20T06:56:06.784Z","v":0}
{...,"req":{"method":"GET","url":"/socket.io/?dataset=Facebook&workbook=4425d4d6a7b26f5a&EI0
{...,"fileName":"graph-viz/viz-server.js","socketID":"9ckImeuIxO_97olrAAAA","level":30,"msg"
```

Send browser instance state

```
{...,"module":"viz-app/worker/services/sendFalcorUpdate.js","level":20,"jsonGraph":{"workboo
{...,"msg":"HTTP request received by Express.js { originalUrl: '/graph/img/logo_white_horiz.
```

Send browser the initial visual graph

```
{...,"module":"viz-app/worker/services/sendFalcorUpdate.js","level":20,"jsonGraph":{"workboo
```

```
{...,"activeBuffers":["curPoints","pointSizes","logicalEdges","forwardsEdgeToUnsortedEdge","
{...,"msg":"CLIENT STATUS true","time":"2018-07-20T06:56:09.861Z","v":0}
{...,"counts":{"num":4039,"offset":0},"msg":"Copying hostBuffer[pointSizes]. Orig Buffer len
{...,"msg":"constructor:  function Uint8Array() { [native code] }","time":"2018-07-20T06:56:
{...,"counts":{"num":176468,"offset":0},"msg":"Copying hostBuffer[logicalEdges]. Orig Buffer
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:56
{...,"counts":{"num":88234,"offset":0},"msg":"Copying hostBuffer[forwardsEdgeToUnsortedEdge]
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:56
{...,"counts":{"num":176468,"offset":0},"msg":"Copying hostBuffer[edgeColors]. Orig Buffer l
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:56
{...,"counts":{"num":4039,"offset":0},"msg":"Copying hostBuffer[pointColors]. Orig Buffer le
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:56
{...,"counts":{"num":8078,"offset":0},"msg":"Copying hostBuffer[forwardsEdgeStartEndIdxs]. O
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:56
{...,"msg":"selectNodesInRect { all: true }","time":"2018-07-20T06:56:09.899Z","v":0}
{...,"msg":"selectNodesInRect { all: true }","time":"2018-07-20T06:56:09.915Z","v":0}
{...,"module":"viz-app/worker/services/sendFalcorUpdate.js","level":20,"jsonGraph":{"workboo
{...,"msg":"CLIENT STATUS false","time":"2018-07-20T06:56:10.088Z","v":0}
{...,"msg":"selectNodesInRect { all: true }","time":"2018-07-20T06:56:10.317Z","v":0}
```

```
{...,"msg":"selectNodesInRect { all: true }","time":"2018-07-20T06:56:10.317Z","v":0}
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo curPoints","time":"2018-07-20T06:56:10.363Z","v":0}
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo pointSizes","time":"2018-07-20T06:56:10.364Z","v":0}
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo forwardsEdgeToUnsortedEdge","time":"2018-07-20T06:56:10
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo logicalEdges","time":"2018-07-20T06:56:10.366Z","v":0}
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo edgeColors","time":"2018-07-20T06:56:10.367Z","v":0}
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo pointColors","time":"2018-07-20T06:56:10.369Z","v":0}
{...,"msg":"selectNodesInRect { all: true }","time":"2018-07-20T06:56:10.371Z","v":0}
```

```
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo forwardsEdgeStartEndIdxs","time":"2018-07-20T06:56:10.6


{...,"msg":"CLIENT STATUS true","time":"2018-07-20T06:56:20.319Z","v":0}
{...,"msg":"CLIENT STATUS false","time":"2018-07-20T06:56:20.413Z","v":0}
```

Run iterative clustering and stream results to client

```
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo curPoints","time":"2018-07-20T06:56:20.837Z","v":0}
{...,"msg":"CLIENT STATUS true","time":"2018-07-20T06:56:25.821Z","v":0}
{...,"msg":"CLIENT STATUS false","time":"2018-07-20T06:56:25.841Z","v":0}
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAAA
{...,"msg":"HTTP GET request for vbo curPoints","time":"2018-07-20T06:56:26.445Z","v":0}


{...,"msg":"CLIENT STATUS true","time":"2018-07-20T06:56:29.099Z","v":0}
{...,"module":"viz-app/worker/services/sendFalcorUpdate.js","level":20,"jsonGraph":{"workboo
```

End session

```
{...,"req":{"method":"GET","url":"/graph/graph.html?dataset=Facebook&workbook=4425d4d6a7b26f
{...,"active":false,"msg":"Reporting worker is inactive.","time":"2018-07-20T06:57:43.135Z",
{...,"msg":"Attempting to exit worker process.","time":"2018-07-20T06:57:43.135Z","v":0}
```

Replacement worker starts as a fresh process / pid

```
{...."msg":"Config options resolved","time":"2018-07-20T06:57:49.471Z","v":0}
...
```

Chapter 23

Some Additional Features for Developers and Sysadmins

Sending a compiled Graphistry distrobution to s3 to install on other systems with 4mo expiray

```
sudo pip install awscli
aws configure
aws s3 cp dist/graphistry.tar.gz s3://airgapped-deploy/graphistry-BUILD-NUMBER.tar.gz
aws s3 presign s3://airgapped-deploy/graphistry-BUILD-NUMBER.tar.gz --expires-in 10368000
```

Download the bundle from s3 with `awscli`

```
aws s3 cp s3://<yourbucket>/graphistry.tar.gz graphistry.tar.gz
```

Download the bundle from s3 with `wget`

```
wget -O graphistry.tar.gz  '<url returned from presign>' # quoting that string is important
```

If you want to get, extract, and bootstrap all in one command:

```
PRESIGN_URL="<url returned from presign>" # quoting that string is important
wget -O graphistry.tar.gz  "${PRESIGN_URL}" && tar -xvf graphistry.tar.gz && ./bootstrap.sh
```

Download the bundle from s3 with `curl`

get_s3_object.sh

```
#!/bin/sh
file=graphistry.tar.gz
bucket=your-bucket
resource="/${bucket}/${file}"
contentType="application/x-compressed-tar"
dateValue="`date +'%a, %d %b %Y %H:%M:%S %z'`"
stringToSign="GET
${contentType}
${dateValue}
${resource}"
s3Key=xxxxxxxxxxxxxxxxxxxx
s3Secret=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
signature=`/bin/echo -en "$stringToSign" | openssl sha1 -hmac ${s3Secret} -binary | base64`
curl -H "Host: ${bucket}.s3.amazonaws.com" \
-H "Date: ${dateValue}" \
-H "Content-Type: ${contentType}" \
-H "Authorization: AWS ${s3Key}:${signature}" \
https://${bucket}.s3.amazonaws.com/${file}
```

```
docker build -t gcli .
# https://forums.docker.com/t/how-can-i-run-docker-command-inside-a-docker-container/337

# Run the CLI container with --net=host to access host networking and mount the docker.sock
nvidia-docker run --net=host -it -v /var/run/docker.sock:/var/run/docker.sock gcli bash

# https://github.com/NVIDIA/nvidia-docker/issues/380
# curl the docker cli REST api before you name the image and somehow docker will launch nvid
docker run -ti --rm `curl -s http://localhost:3476/docker/cli` nvidia/cuda nvidia-smi



# https://stackoverflow.com/questions/22944631/how-to-get-the-ip-address-of-the-docker-host-
export HOST_MACHINE_ADDRESS=$(/sbin/ip route|awk '/default/ { print $3 }')
docker run -ti --rm `curl -s http://$HOST_MACHINE_ADDRESS:3476/docker/cli` nvidia/cuda nvidi
```

Chapter 24

Investigation Templates

Investigation templates bring a lightweight form of automation to investigations. They work just like regular investigations, except they add a few key features that, combined with existing investigation features, unlock useful workflows.

Contents 1. Sample workflows 1. Create a template 1. Manual: Instantiate a template 1. URL API: Linking a template 1. Splunk integration 1. Best practices * Manual data for first step * Multiple entry points * Set time range and provide instructions * Naming * Cross-linking

1. Sample workflows

- **In-tool**: Create a base template such as for looking at an account, and instantiate whenever you are investigating a new account
- **From an alert email or dashboard**: Include a link to a 360 view for that alert or involved entities, and center it on the time range of the incident
- **Splunk UI**: Teach Splunk to include 360 views whenever it mentions an account, IP, or alert

2. Create a template

Any investigation can be reused as a template. From an investigation (or `save-a-copy` of one), in the investigation details, check `Template`. When you save and return to the content home, it should have moved into the top `Templates` section.

3. Manual: Instantiate a template

From the content home, navigate to your template, and press the `new` button. This will create a new investigation that is based off of the most recent version of the template, similar to how `clone` works on an investigation. Editing a template keeps past investigations safe and untouched.

4. URL API: Linking a template

The magic happens when the URI API is used to enable users of web applications to jump into prebuilt investigations with just one click.

Consider the following URL for triggering a phone history check:

```
/pivot/template?investigation=453d190914cf9fa0&pivot[0][events][0][phone]=1.800.555.5555&tim
```

This URL: Instantiates template `453d190914cf9fa0`, names it `Phone-History-555-5555`, overrides the global time range to center at `1504401120` (epoch time) and runs searches +/- 1 day from then. The first pivot will be populated with one record, and that record will have field `phone` mapped to the string `"1.800.555.5555"`.

FIELD	OPTIONAL	DEFAULT	FORMAT	NOTES
investigation	required		ID	Get template ID from its URL. Ex: 453d190914cf9fa0
name	optional	"Copy of [template name]"	String	Recommend using a short standard pattern to group together ("[Phone History] . . . ")

FIELD	OPTIONAL	DEFAULT	FORMAT	NOTES
time	optional	now	Number or string	Epoch time (number) or best-effort if not a number. Ex: 1504401120
before	optional	-7d	[+/-][number][ms/s/min/h/d/w/mon/y]	Ex: -1d
after	optional	+0d	[+/-][number][ms/s/min/h/d/w/mon/y]	Ex: +3min
pivot	optional		see below	see below

URL parameter `pivot` follows one of the two following formats: * `[step][field]`, e.g., `pivot[0][index]=index%3Dalerts`, the URI-encoded form of string `"index=alerts"` * `[step][field][list_index][record_field]`, e.g., `pivot[0][events][0][phone]=1-800-555-5555` sets the first step's events to JSON list `[ {"phone": "1-800-555-5555"} ]`

You can therefore set or override most investigation step values, not just the first one. Likewise, if you want to trigger an investigation over multiple values, you can provide a list of them.

5. Splunk integration

Splunk users can easily jump into Graphistry investigations without much thinking from any Splunk search result or dashboard, even if they don't know which ones are available ahead of time. To do so, you simply register Graphistry templates as Splunk workflow actions.

To make a template appear as a Workflow Action on a specific kind of event:

1. Settings -> Event Types -> new:

   - Search string: The events you want the template to appear on (if you don't hav event types already known). Ex: "index=calls phone=*“.
   - Tag(s): An identifier to associate with these events

2. Settings -> Fields -> Workflow actions -> new

- Label: What appears in Splunk's action menu. Ex: `Check Graphistry for Phone 360: $phone$`
- Apply only to fields, tags: the search result column and/or tag from Step 1
- Show action in: Both
- Action type: Link
- Link configuration: Template URL, using `$fld$` to populate values. Ex: `https://my_graphistry.com/pivot/template?investigation=453d190914cf9fa0&pivot[0`
- Open link in: New window
- Link method: get

6. Best practices

Manual data for first step

By making the first step an `Enter data` one, most of the parameters can be set on it. The URL generates an initial graph, and subsequent steps expand on them.

Multiple entry points

You can likely combine multiple templates into one. For example, in IT scenarios, 360 views for IP's, MAC addresses, and host names likely look the same. Make the first step create a graph for one or more of these, the next ones derive one value type from the other (or a canonical ID), and the remaining steps look the same.

Set time range and provide instructions

Analysts unfamiliar with your template would strongly benefit from instructions telling them what to modify (if anything) and how to use the investigation. Many options likely have sane defaults on a per-template basis, such as the time range, so we recommend including them in your URLs.

Naming

Content management can become an issue. Use a custom short description name, such as `name=%5BPhone%20360%5D%20555-5555` (=> `[Phone 360] 555-5555`. The generated investigations can now be easily searched and sorted.

Cross-linking

You can include templates as links within templates! For example, whenever a phone number node is generated, you can include attribute `link` with value `/pivot/template?investigation=...` .

Chapter 25

Manual inspection of all key running components

Takes about 5-10min

0. Start

- Put the container in `/var/home/my_user/releases/my_release_1`: Ensures relative paths work, and good persistence hygiene across upgrades
- Go time!

  ```
  docker load -i containters.tar
  docker-compose up
  ```

1. Static assets

- Go to http://graphistry
- Expect to see something similar to http://labs.graphistry.com
- Good way to check for TLS and container load failures

2. Visualization of preloaded dataset

- Go to http://graphistry/graph/graph.html?dataset=Facebook
- Can also get by point-and-clicking if URL is uncertain: http://graphistry -> `Learn More` -> (the page)

- Expect to see something similar to http://labs.graphistry.com/graph/graph.html?dataset=Facebook
- If points do not load, or appear and freeze, likely issues with GPU init (driver) or websocket (firewall)
- Can also be because preloaded datasets are unavailable: not provided, or externally mounted data sources
    - In this case, use ETL test, and ensure clustering runs for a few seconds (vs. just initial pageload)

3a. Test `/etl` and PyGraphistry

Do via notebook if possible, else `curl`

- Get API key by running from host:

```
docker-compose exec central curl -s http://localhost:10000/api/internal/provision?text=MYUSE
```

- Install PyGraphistry and check recent version number (Latest: https://pypi.org/project/graphistry/)

```
!pip install graphistry -q
import graphistry
graphistry.__version__
```

- Try your key, will complain if invalid, otherwise silent

```
graphistry.register(protocol='http', server='my.server.com', key='my_key')
```

- Try upload and viz, may need to open result in new tab if HTTPS notebook for HTTP graphistry. Expect to see a triangle:

```
import pandas as pd
df = pd.DataFrame({'s': [0,1,2], 'd': [1,2,0]})
graphistry.bind(source='s', destination='d').plot(df)
```

3b. Test `/etl` by commandline

If you cannot do **3a**, test from the host via `curl` or `wget`:

- Make `samplegraph.json`:

```
{
    "name": "myUniqueGraphName",
    "type": "edgelist",
    "bindings": {
        "sourceField": "src",
        "destinationField": "dst",
        "idField": "node"
    },
    "graph": [
      {"src": "myNode1", "dst": "myNode2",
       "myEdgeField1": "I'm an edge!", "myCount": 7},
      {"src": "myNode2", "dst": "myNode3",
        "myEdgeField1": "I'm also an edge!", "myCount": 200}
    ],
    "labels": [
      {"node": "myNode1",
       "myNodeField1": "I'm a node!",
       "pointColor": 5},
      {"node": "myNode2",
       "myNodeField1": "I'm a node too!",
       "pointColor": 4},
      {"node": "myNode3",
       "myNodeField1": "I'm a node three!",
       "pointColor": 4}
    ]
}
```

- Get API key

```
docker-compose exec central curl -s http://localhost:10000/api/internal/provision?text=MYUS
```

- Run ETL

```
curl -H "Content-type: application/json" -X POST -d @samplegraph.json https://labs.graphistr
```

- From response, go to corresponding http://graphistry/graph/graph.html?dataset=...
- check the viz loads
- check the GPU iteratively clusters

4. Test pivot

4a. Basic

- Test it loads at http://graphistry/pivot
- Connector page only shows WHOIS and HTTP pivots (http://graphistry/pivot/connectors), and clicking them returns green

4b. Investigation page

- Starts empty at http://graphistry/pivot/home
- Pressing + creates a new untitled investigations

- Can create and run a manual pivot in it, with settings: “‘ Pivot: Enter data Events: [{ “x”: 1, “y”: “b”}] Nodes: x y
- Expect to see a graph with 1 event node, and 2 connected entity nodes `1` and `b` “‘

4c. Configurations

- Edit `.env` and `docker-compose.yml` as per below
- Set each config in one go so you can test more quickly, vs start/stop.
- Run

  ```
  docker-compose stop
  docker-compose up
  ```

4c.i Password

- Edit `.env` to uncomment `PIVOT_PASSWORD=something`
- Going to http://graphistry/pivot should now challenge for `graphistry` / `something`

4c.ii Persistence

- Pivot should persist to `./data` already by default, no need to do anything
- Edit `docker-compose.yml` to uncomment `viz`’s `volume` persistence mounts for `./data`
- Run a pivot investigation and save: should see `data/{investigation,pivot,workbook_cache,data_cach`

4c.iii Splunk

- Edit `.env` for `SPLUNK_HOST`, `SPLUNK_PORT`, `SPLUNK_USER`, `SPLUNK_KEY`
- Run one pivot:

  ```
  Pivot: Search: Splunk
  Query: *
  Max Results: 2
  Entities: *
  ```

- Expect to see two orange nodes on the first line, connected to many nodes in the second

4c.iv Neo4j

- Edit `.env` for `NEO4J_BOLT` (`bolt://...:...`), `NEO4J_USER`, `NEO4J_PASSWORD`
- Test status button in http://graphistry/pivot/connectors
- Make a new investigation
- Pivot 1

  ```
  Pivot: Search: Neo4j
  Query: MATCH (a)-[e*2]->(b) RETURN a,e,b
  Max Results: 10
  Entities: *
  ```

- Pivot 2

  ```
  Pivot: Expand: Neo4j
  Depends on Pivot 1
  Max Results: 20
  Steps out: 1..1
  ```

- Run all: Gets values for both

4c. ELK, VT: Later

5. Test TLS Certificates

AWS: * In EC2: Allocate an Elastic IP to your instance (may be optional) * In Route53: Assign a domain to your IP, ex: `mytest.graphistry.com` * If needed, run `DOMAIN=my.site.com ./scripts/letsencrypt.sh` and `./gen_dhparam.sh` * Follow `docker-compose.yml` instructions to enable: * In `graphistry.conf` (pointed by `docker-compose.yml`), uncomment `ssl.conf` include on last line * Restart, check pages load * Try a notebook upload with `graphistry.register(...., protocol='https')`

Chapter 26

Threat Model

Graphistry is largely a standard enterprise webapp and uses modern design patterns, infrastructure, tools, & frameworks.

Interesting surface areas include: use of GPUs, Jupyter notebooks, and the distinctions between authenticated users (privileged analyst teams) vs. network users (shared visualization recipients.)

Interesting infrastructure and controls include: Docker containers & networking & volumes, Nginx routing, and Django auth modules.

The Embedding API is out of scope for this document.

Assets

- System
- Connector config
- Authored investigations + templates + visualizations
- Notebooks

Role hierarchy with asset access levels (read/write)

- Admins: DB connector config
- Analyst team: cases/templates/notebooks
- Network user: visualizations

Authentication

- Web access: pluggable web auth (nginx/django)
- OS access: owner-controlled; recommend firewall restricts to http/https/ssh

Authorization:

- Admins: OS access secured by owner (recommend: firewall + SSH key)
- Analyst: Web login (enabled by admin), all analysts share web-based investigations & automations & notebooks
- Network user: Generated visualizations shared via web keys with any network-connected user, with options for read-only and read+write

Attack surfaces:

- HW+OS: Out of scope
- Supply chain: Delivered binary & packaged dependencies
- Logs
- Web auth
- Authenticated user: All web routes
- Authenticated user: Notebooks, which exposes notebook data volume mount and allows arbitrary code in the (restricted) notebook container
- Network user: Access to viz service and volume mounts
- Individual tools & frameworks, especially Docker, Nginx, NodeJS/Fastify/Express, Python, Nvidia RAPIDS, & Jupyter

Architecture: Defense-in-depth & trust boundaries

- Dependencies are explicitly versioned, and regularly updated based on community scan warnings (npm audit, docker, . . .)
- Software delivered via signed AWS S3 URL or cloud AMI/Marketplace
- Config: App reads from environment variable or config mounts. Explicit schema tags sensitive values, and app respects those tags when emitting to logs or the UI.
- Isolated docker services with configured volume mounts: Resources are physically seperated, including limiting which mounts are exposed to the services exposed to network users vs. authenticated app regions.
- Nginx container controls routes, including enforcing auth on public routes

- Service runtimes are primarily in managed languagues that enforce memory isolation & additional process isolation
- Where the app does support providing code, approach taken of either whitelisting (e.g., client query parameters), and app-level or ephemeral interpreters (vs. reusing persistent DBs)
- HTTP activity is logged