Contents

Chapter 1

Welcome to Graphistry: Admin Guide

Graphistry is the most scalable graph-based visual analysis and investigation automation platform. It supports both cloud and on-prem deployment options. Big graphs are tons of fun!

Docs

This documentation cover system administration. See the **Further reading** section at the bottom of this page for the list of top-level administration guides. For analyst and developer guides, see the main docs accessible from the Graphistry web UI: `your_graphistry.acme.ngo/docs`.

Support

Email, Zoom, Slack, phone, tickets – we encourage using the Graphistry support channel that works best for you. We want you and your users to succeed!

Quick launch

The fastest way to start using Graphistry is to quick launch a private preconfigured Graphistry instance from the AWS and Azure marketplaces. They run within your cloud provider account and therefore facilitate experimentation with real data. These docs will still be helpful for advanced configuration and maintenance. See AWS launch walkthrough tutorial & videos.

Advanced administration

The admin guide covers:

- Deployment planning:
- Architecture selection: Hosted, cloud marketplaces, cloud, on-prem, hybrid, and air-gapped
- Capacity planning and software requirements
- Installation & testing
- Configuration
- Operation
- Backup, maintenance, and upgrades

Cloud Marketplace Admins: You can focus exclusively on sections on capacity planning, optional configurations, and optional maintenance, . Congrats!

Manul Install Admins: The use of GPU computing makes administration a bit different than other software. Graphistry ships batteries-included to make operations close to what you'd expect of modern containerized software. However, this includes setup of Nvidia drivers, Docker, docker-compose, and the nvidia-docker runtime.

Top commands

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

Login to server:

- AWS: `ssh -i [your_key].pem ubuntu@[your_public_ip]`
- Azure: `ssh -i [your_key].pem [your_user]@[your_public_ip]` or `ssh [your_user]@[your_public_ip]` (pwd-based)
- On-prem / BYOL: Contact your admin

TASK	COMMAND	NOTES
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`	Start Graphistry, close with ctrl-c
Start daemon	`docker-compose up -d`	Start Graphistry as background process

TASK	COMMAND	NOTES
Start	`docker-compose -p my_namespace up`	Start Graphistry under a unique name (in case of multiple versions). NOTE: must modify volume names in `docker-compose.yml`.
Stop	`docker-compose stop`	Stop Graphistry
Restart	`docker restart <CONTAINER>`	
Status	`docker-compose ps`, `docker ps`, and `docker status`	Start-Up-time, healthchecks, ...

TASK	COMMAND	NOTES
API Key	docker-compose exec streamgraph-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`

TASK	COMMAND	NOTES
Reset	`docker-compose down -v && docker-compose up`	Stop Graphistry, remove all internal state (including user accounts), and start fresh.
Create Users	Use Admin Panel (see Create Users)	

Manual enterprise install

NOTE: Managed Graphistry instances do not require any of these steps.

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

Manual environment setup

See sample Ubuntu 18.04 TLS and RHEL 7.6 environment setup scripts for Nvidia drivers, Docker, nvidia-docker runtime, and docker-compose.

Please contact Graphistry staff for environment automation options.

Manual Graphistry container download

Download the latest enterprise distribution from the enterprise release list on the support site. Please contact your Graphistry support staff for access if not available.

Manual Graphistry container installation

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
```

Further reading

- Plan a deployment:
- Architecture: Deployment architecture planning guide
- Capacity: Hardware/software requirements
- Install:
-
- Manual Graphistry Installation for non-marketplace / non-hosted: AWS BYOL, Azure BYOL, On-Prem (RHEL/Ubuntu), & Air-Gapped
- Configure
- Configure main system: TLS/SSL/HTTPS, backups to disk, multiple proxy layers, and more
  - AWS Marketplace quickstart
  - Azure Marketplace quickstart
- Configure investigations and the Graphistry Data Bridge: Connectors, ontology, automations, and more
- Configure users
- Configure PyGraphistry and notebooks
- Maintain
- On unconfigured instance reboots, you may need to first run `sudo systemctl start docker`
- Updates, backups, and migrations
- Test:

- Testing an install

Chapter 2

Recommended Deployment Configurations: Client, Server Software, Server Hardware

The recommended server configuration is AWS Marketplace and Azure Marketplace with instance types noted on those screens. The one-click launcher deploys a fully preconfigured instance.

Graphistry Enterprise also ships as a Docker container that runs in Linux Nvidia GPU environments that are compatible with NVIDIA RAPIDS and the Nvidia Docker runtime. It is often run in the cloud as well.

Contents

Overview

- **Client**: Chrome/Firefox from the last 3 years, WebGL enabled (1.0+), and 100KB/s download ability

- **Server**:
- Minimal: x86 Linux server with 4+ CPU cores, 16+ GB CPU RAM (3GB per concurrent user), 150GB+ disk, and 1+ Nvidia GPUs (Pascal onwards for NVIDIA RAPIDS) with 4+ GB RAM each (1+ GB per concurrent user)
- Recommended: Ubuntu 18.04 LTS, 4+ CPU cores, 64GB+ CPU RAM, 150GB+ disk, Nvidia Pascal or later (Volta, RTX, ...) with 12+GB GPU RAM
- CUDA driver rated for NVIDIA RAPIDS
- Nvidia Docker runtime set as default runtime for docker-compose 1.20.0+ (yml file format 3.4+)

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
- **WebGL**: WebGL 1.0 is required. It is 7+ years old, so most client devices, including phones and tablets, support it: check browser settings for enabling. 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:

- P3.2 ***Recommended for testing and initial workloads***

Tested Azure Instances:

- NC6s_v2 ***Recommended for testing and initial workloads***
- NC6s_v3

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

Please contact for discussion of multi-GPU scenarios.

OS & Docker

Graphistry runs preconfigured with a point-and-click launch on Amazon Marketplace.

Graphistry regularly runs on:

- Ubuntu Xenial 16.04+ LTS ***Recommended***
- RedHat RHEL 7.4+

Both support Nvidia / Docker:

- CUDA driver rated for NVIDIA RAPIDS
- Nvidia Docker *native* runtime (for after Docker 19.03)
- docker-compose 1.20.0+ (yml file format 3.4+) with default runtime set as `nvidia` at time of launch

See Ubuntu 18.04 LTS manual configuration and RHEL 7.6 manual configuration for an example of setting up Nvidia for containerized use on Linux. Marketplace deployments come preconfigured with the latest working drivers and security patches.

User: Root vs. Not, Permissions

Installing Docker and Nvidia dependencies currently require root user permissions.

Graphistry can be installed and run as an unprivileged user as long as it has access to Docker and the Nvidia runtime.

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

Graphistry requires NVIDIA RAPIDS-compatible GPUs. The following GPUs, Pascal and later, are known to work with Graphistry:

- P100, V100, RTX
- ... Found both in DGX and DGX2

The GPU should provide 1+ GB of memory per concurrent user. At 4GB of GPU RAM is required, and 12GB+ is recommended.

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 and can vertically scale to multiple CPUs+GPUs on the same node for additional users.

- 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.

HA

Graphistry resiliency typically comes in multiple forms:

- User separation: For larger deployments, we recommend separating developers (unpredictable), power users (many large graphs), and regular users (many small sessions).
- Physical resource isolation: Graphistry can run on the same device as other software as long as they respect Graphistry's CPU, GPU, and network resources. You can use Docker to limit Graphistry execution to specific CPU, GPU, and network resources. Check your other software to ensure that it can likewise be configured to not interfere with sibling workloads.
- Process isolation: You may run multiple Graphistry instances on the same node to increase resiliency between groups of users. This can be combined with same-node physical resource isolation. This increases resiliency up to hardware and driver failure. However, note that each Graphistry instance will consume 1.5GB+ of GPU RAM.
- Logical separation & replication: You may want to further tune software replication factors. Graphistry runs as multiple containerized shared services with distinct internal replication modes and automatic restarts. Depending on the service, a software failure may impact live sessions of cotenant users or prevent service for 3s-1min. Within a node, you may choose to either tune internal service replication or run multiple Graphistry instances.
- Safe upgrades: Due to Graphistry's use of version-tagged Docker images and project-namespaceable docker-compose orchestrations, upgrades can be performed through:
- New instances (e.g., DNS switch): recommended, especially for cloud

- Installation of a concurrent version Contact support staff for migration information.

Chapter 3

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 4

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 5

2. Configure instance

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

Chapter 6

3. General installation

Proceed to the instructions for general installation.

Chapter 7

Graphistry in AWS Marketplace

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

Instance Type

Pick a `P3.2+` GPU instance in your region. Graphistry requires a Nvidia RAPIDS compatible GPU, whose minimal requirement is a Pascal-generation GPU or later.

Basic administration

- Create users
- Generate API keys for individuals without accounts
- Turn server on-and-off via AWS Console via **stop** and **start**
- Advanced configuration
- Update, backup, and migrate
- To simplify administration and limit downtime, we recommend creating a new Marketplace instance, copying data snapshots to it and loading it in, and switching DNS to the new instance only when tested

Common marketpace 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

- Associate your AWS instance with an Elastic IP or a domain
- Setup TLS
- In restricted environments, constrain networking to a safelist, e.g., VPN, and optional, change logging drivers to stop Graphistry from recieving maintenace logs

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. Command-line Login

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 the AWS Marketplace AMI preconfigures the `nvidia` runtime for `docker`.

```
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
```

Note: Precise set of containers changes across versions

5. Install Python packages

If you see `wheel` errors, you may need to run `pip 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 by logging into 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, and will delete all users, but no data.

I want to log into the server

See section `login`

See general installation for further information.

Chapter 8

Graphistry on Azure: Manual Environment Setup Instructions

DEPRECATION WARNING:

Get started more quickly and securely with Graphistry in Azure Marketplace.

We no longer recommend manually installing drivers via the original Graphistry-maintained bootstrap scripts. Instead, we now recommend using Graphistry in Azure Marketplace which has been preconfigured, and for advanced manual enterprise users, to use the Nvidia GPU Container (Ubuntu) "Nvidia NGC" base image.

Chapter 9

Deprecated 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

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

- **Minimal GPU type**: NC6v2 (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 `NCv2+ Series` and `ND+ 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., NC6v2 (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`

Install docker-compose:

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.23.1/docker-compose-$(un
sudo chmod +x /usr/local/bin/docker-compose
```

NGC already sets the default docker runtime to nvidia for you (`/etc/docker/daemon.json`).

From here, you can perform a general installation.

Chapter 10

Graphistry Data Bridge for Proxying

Graphistry supports bridged connectors, which eases tasks like crossing from a cloud server to on-prem databases. It is designed to work with enterprise firewall policies such as HTTP-only and outgoing-only. Instead of Graphistry directly making requests against API servers, Graphistry sends the requests to the data bridge server, which then proxies the query and sends the results back. Graphistry can run a mixture of direct and bridged connectors.

Prerequisites

- Standard Graphistry GPU application server (ex: cloud), with admin access
- Data bridge docker container. You can find `bridge.tar.gz` in your distribution's release bundle and, for managed Graphistry users, by logging into the instance and scp'ing `/home/ubuntu/graphistry/bridge.tar.gz`.
- Server to use as a bridge (typically on-prem), with admin access
- CPU-only OK
- Linux:`docker` and `docker-compose`
- Firewall permissions between DB <> bridge and bridge <> Graphistry

Updates

Starting with `2.23.0`, you can use old bridge server versions with new Graphistry distributions. So updating your Graphistry application server, unless otherwise stated in the release note history, should keep working with your existing bridge

server. The bridge server will automatically reconnect to the `GRAPHISTRY_HOST` its `connector.env` points to.

Architecture

Graphistry GPU application server

- Bridge-mode connectors: Built into standard Graphistry servers
- Individual database connectors can be configured to use a bridged connector instead of a direct API connector

Graphistry data bridge server

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

Key configuration

- For each connector, set unguessable UUID strings for the server and client - this enables either side to trust and revoke access. These go in the bridge's `connector.env` and the main application server's `.env`
- Revocation can occur by server or client due to use of 2 keys

Example: Splunk

Generate a key

- Can be any string
- Ex: Unguessable strings via `docker-compose exec pivot /bin/bash -c "../../../node_modules/uuid/bin/uuid v4"` => `<my_key_1>`

Graphistry GPU application server

- Configure: In `.env`, setup the Splunk connector as usual and then set `SPLUNK_USE_PROXY` and the two keys to proxy through a data bridge server:

```
### Splunk connector config
SPLUNK_HOST=splunk.example.com
SPLUNK_WEB_PORT=3000
SPLUNK_USER=my_user
```

```
SPLUNK_KEY=my_pwd

### Splunk bridge settings
SPLUNK_USE_PROXY=true
SPLUNK_SERVER_KEY=my_key_1
SPLUNK_PROXY_KEY=my_key_2
```

- Launch: Restart Graphistry via `docker-compose stop pivot && docker-compose up -d pivot`.

The connector will start looking for the data bridge.

- Test: In the connector settings panel of `/pivot/home`, click the `check status` button for the connector. It should report success and turn green if it successfully exchanges keys with your bridge and runs a proxied connector command

Graphistry data bridge server

Edit `connector.env`:

```
#REQUIRED: Fill in with your server
GRAPHISTRY_HOST=https://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
```

If your Graphistry instance uses a non-standard prefix, set `GRAPHISTRY_MOUNT_POINT=/my_subdir`

Install and launch data bridge

1. Install

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

2. Configure

Edit `.env`. See above example.

3. Launch

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

The data bridge will autoconnect to your Graphistry application server.

4. Test

See `test` section for the Graphistry GPU application server.

5. Debug

Enable the below in the data bridge's `connector.env` and restart it:

```
DEBUG=*
GRAPHISTRY_LOG_LEVEL=TRACE
```

Watch your bridge's logs and your app server's logs: `docker-compose logs -f -t --tail=1`

Chapter 11

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 12

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 13

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 14

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/em
        "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:
            { "myType": "car", ... }`,
        format: Object,
        default: {},
        arg: 'icons',
        env: 'GRAPHISTRY_ICONS'
    },
    colors: {
        doc: `JSON dictionary from entity type to color hex code:
            { "myType": "#FF0000", ... }`,
        format: Object,
        default: {},
        arg: 'colors',
        env: 'GRAPHISTRY_COLORS'
    },
    sizes: {
        doc: `JSON dictionary from entity type to size integers (1-1000), with Graphistr
            { "myType": 100, ... }`,
        format: Object,
        default: {},
        arg: 'sizes',
        env: 'GRAPHISTRY_SIZES'
    },
    products: {
        doc: `JSON list of per-product dictionaries:
          [{"name": "myProduct1",
            ? productIdentifier: {"field1": "value1", ...}, // index selector
            ? fieldsBlacklist: [ "field1", ... ], //exclude from data extraction
            ? attributesBlacklist: [ "field1", ... ], //exclude from entity drilldown
            ? entitiesBlacklist: [ "field1", ... ], //exclude from generated nodes
            ? defaultFields: [ "field1", ... ], //populate dropdowns
            ? desiredEntities: [ "field1", ...], //default nodes
            ? desiredAttributes: [ "field1", ...], //default drilldowns
            ? colTypes: { "col1": "type1", ... } //what node type to generate from a col
        }, ...]`,
        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 15

Configure PyGraphistry

PyGraphistry can be configured for tasks such as eliminating analyst boilerplate, customizing security handling, and supporting advanced server configurations.

Configuration is for two basic APIs: the upload API for how PyGraphistry sends data to a Graphistry server, and the client API for how a URL is loaded into a user's browser.

For server configuration, see main configuration docs. For REST API settings for uploading and viewing visualizations, see main developer documentation.

The Cascade

PyGraphistry configuration settings resolve through the following cascade, with the top items overriding the further ones:

Priority	Name	Primary Use
5	`graph.settings`	Ad-hoc or developer fine-tuning an individual visualization's style via `graph.settings(url_params={...})`
4	`graphistry.register(...)`	Analyst or developer
3	Environment variables	Developer or sysadmin
2	`graphistry.config`	Sysadmin
1	Default	

Graphistry's built-in Jupyter server comes with a predefined `graphistry.config`.

Settings

Setting	Default	Type	Description
`api`	1	`1` *JSON* `2` *protobuf* `3` (contact: Arrow)	Upload format and wire protocol
`api_key`	None	string	
`certificate_validation`	True	boolean	Disable to allow ignore TLS failures such as for known-faulty CAs

Setting	Default	Type	Description
`client_protocol`	None	FQDN	hostname including protocol, for overriding `protocol` and `hostname` for what URL is used in browsers from displaying visualizations
`dataset_upload_prefix`	`"Pyodide/"`	String	Prefix on upload location

Setting	Default	Type	Description
`hostname`	`"hub.graphistry.com"`	String	Domain (and optional path) for where to upload data and load visualizations
`protocol`	`"https"`	`"https"` or `"http"`	

Usage Modes

graph.settings(url_params={...})

Override and add query parameters to the loaded visualization iframe by adding key/value strings to `url_params`. This does not control the protocol, domain, nor path, so is primarily for styling and debugging purposes.

Example:

```
my_graph.settings(url_params={
    'play': '0',
    'my_correlation_id': 'session-123'
}).plot()
```

graphistry.register()

Note: Setup of the environment lets developers and analysts skip manually configuring `register()`, which may be preferrable

Global module settings can be defined via `register()`:

```
register(key=None, server=None, protocol=None, api=None, certificate_validation=None, ...)
```

See PyGraphistry docs for individual connectors such as `.bolt(...)` and `.tigergraph(...)`.

Example: Neo4j (bolt/cypher)

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

g = graphistry.bolt({'server': 'bolt://...', 'auth': ('my_user', 'my_pwd')})

g.cypher("MATCH (a)-[b]->(c) RETURN a,b,c LIMIT 10").plot()
g.cypher("MATCH (a)-[b]->(c) RETURN a,b,c LIMIT 100000").plot()
...
```

Environment variables

Graphistry Setting	Environment Variable	Description
`api_key`	`GRAPHISTRY_API_KEY`	
`api_version`	`GRAPHISTRY_API_VERSION`	
`certificate_validation`	`GRAPHISTRY_CERTIFICATE_VALIDATION`	
`client_protocol_hostname`	`GRAPHISTRY_CLIENT_PROTOCOL_HOSTNAME`	
`dataset_prefix`	`GRAPHISTRY_DATASET_PREFIX`	
`hostname`	`GRAPHISTRY_HOSTNAME`	
`protocol`	`GRAPHISTRY_PROTOCOL`	
	`PYGRAPHISTRY_CONFIG`	Absolute path of `graphistry.co`

There are multiple common ways to set environment variables:

- OS user login, e.g., `.bashrc` file
- In an invocation script, `MY_FLD=MY_VAL python myscript.py`
- The `environment:` section of Graphistry's `~/docker-compose.yml` for `notebook` or the `~/.env` file. WARNING: Editing `.env` is preferred over editing `.yml` in order to simplify upgrading
- In Python via `os.environ['MY_FLD'] = 'MY_VAL'`

graphistry.config

Specify a `json` file using key/values from the Settings table.

PyGraphistry automatically checks for `graphistry.config` as follows:

```
config_paths = [
    os.path.join('/etc/graphistry', '.pygraphistry'),
```

```
    os.path.join(os.path.expanduser('~'), '.pygraphistry'),
    os.environ.get('PYGRAPHISTRY_CONFIG', '')
]
```

If you are using Graphistry's built-in Jupyter server, it autoconfigures `PYGRAPHISTRY_CONFIG` and `graphistry.config`. Modify `docker-compose.yml`'s `notebook` section's volume mounts for new behavior. You can check the container's default config by running `! cat /home/graphistry/pygraphistry.config` from a notebook.

Examples

Speed up some uploads

```
import graphistry
graphistry.register(api=2)
```

Preset an API key for all system users

Create Python-readable `/etc/graphistry/.pygraphistry`:

```
{
    "api_key": "SHARED_USERS_KEY",
    "protocol": "https",
    "hostname":"my.server.com"
}
```

For Jupyter notebooks, you may want to create per-user login `.pygraphistry` files. Please contact staff for further options and requests.

Different URLs for internal upload vs external viewing

```
import os
import graphistry

### Internal URL: PyGraphistry app -> Graphistry Server
GRAPHISTRY_HOSTNAME_PROTOCOL='https'
os.environ['GRAPHISTRY_HOSTNAME'] = "10.0.0.20"

### External URL: Webpage iframe URL -> Graphistry Server
os.environ['GRAPHISTRY_CLIENT_PROTOCOL_HOSTNAME'] = "https://graph.site.ngo/graphistry"

graphistry.register(
    key='MY_API_KEY',
```

```
    protocol=GRAPHISTRY_HOSTNAME_PROTOCOL
)
```

Chapter 16

Configuring Graphistry

Administrators can add users, 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.
- See update, backup, & migration instructions for preserving configurations and data across installations.
- For advanced Python notebook and application configuration, see PyGraphistry configuration.

Add your team and provide API keys

See user creation docs

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.
- Custom TLS is via editing `Caddyfile`(Caddy docs) and mounting your certificates via `docker-compose.yml` (Caddy Docker docs). Caddy supports LetsEncrypt with automatic renewal, custom certificates and authorities, and self-signed certificates. Deprecated, you can also modify Nginx config (`etc/ssl/*`)

Setup free Automatic TLS

Caddy supports free automatic TLS as long as your site meets the listed conditions.

Sample `Caddyfile`:

```
https://*.website.org:443 {
  tls {
    max_certs 100
  }
  proxy / nginx:80 {
    websocket
  }
}
```

Setup custom certs

- Place your certs in `./.caddy/my.crt` and `./.caddy/my.key`
- Modify `Caddyfile`:

```
https://your.site.ngo:443 {
  tls /root/.caddy/my.crt /root/.caddy/my.key
  proxy / nginx:80 {
    websocket
  }
}
```

Note the use of a fully qualified domain name in the first line, and that the file paths are for the `caddy` container's file system, not the host file system

Debugging TLS

Custom TLS setups often fail due to the certificate, OS, network, Caddy config, and file permissions. To perform isolated checks on each, try:

- Test the certificate
- Test a standalone Caddy static file server
- ... Including on another box, if OS/network issues are suspected
- Check the logs of `docker-compose logs -f -t caddy nginx`
- Test whether the containers are up and ports match via `docker-compose ps`, `curl`, and `curl` from within a docker container (so within the docker network namespace)

If problems persist, please reach out to your Graphistry counterparts. Additional workarounds are possible.

Connectors

Optionally, you can configure Graphistry to use database connectors. Graphistry will orchestrate cross-database query generation, pushing them down through the database API, and returning the combined results to the user. This means Graphistry can reuse your existing scaleout infrastructure and make its data more accessible to your users without requiring a second copy to be maintained. Some connectors further support use of the Graphistry data bridge for proxying requests between a Graphistry cloud server and an intermediate on-prem data bridge instead of directly connecting to on-prem API servers.

Security Notes

- Graphistry only needs `read only` access to the database
- Only one system-wide connector can be used per database per Graphistry virtual server at this time. Ex: You can have Splunk user 1 + Neo4j user 2 on one running Graphistry container, and Splunk user 3 + Neo4j user 2 on another running Graphistry container.
- Templates and other embedding modes do not require further Graphistry configuration be beyond potentially API key generation. However, Graphistry implementors will still need access to external dashboards, APIs, etc., into which they'll be embedding Graphistry views.

Get started

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

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

2. Restart `graphistry`, or at least the `pivot` service:

`docker-compose restart` or `docker-compose restart pivot`

3. Test

- In `/pivot/connectors`, configured databases should appear in the live connectors section, and clicking the status check should turn them green
- Running a sample investigation with a database query should return results

Example: Splunk

1. Create a restricted Splunk API user role from the Splunk Web UI

- `Settings` -> `Users` -> `Add new`
- Name: any, such as `graphistry_role`
- For `capabilities`: `rest_properties get`, `rtsearch`, `search`
- For `indexes`: Any that you want exposed to the investigtation team

2. Create a restricted Splunk API user from the Splunk Web UI

- `Settings` -> `Users` -> `Add new`
- Record their name/pwd
- Assign them to the role `graphistry_user` from step 1

3. Configure Graphistry's `.env` with the Splunk server and user information:

```
### Required
#SPLUNK_HOST=splunk.acme.org
#SPLUNK_USER=admin
#SPLUNK_KEY=...

### Optional
#SPLUNK_SCHEME=https
#SPLUNK_PORT=8089
#SPLUNK_WEB_PORT=443
#SPLUNK_SUFFIX=/en-US
#SPLUNK_CACHE_TIMEOUT=14400
#SPLUNK_SEARCH_MAX_TIME=20
#SPLUNK_USE_PROXY=false
#SPLUNK_PROXY_KEY=...
#SPLUNK_SERVER_KEY=...
```

4. Restart and test the connector as per above

Data bridge

In scenarios such as a Graphistry cloud server accessing on-prem API servers, an intermediate on-prem data bridge may be necessary. You can mix bridged and direct (default) connectors in the same Graphistry server. Learn more about the Graphistry data bridge.

Variants

- Give your Graphistry implementation user increased permissions so they can embed Graphistry into existing dashbboard and notification systems, such as for embedded visualizations and quicklinks into contextual investigation templates* Run a Graphistry data bridge, if available for your connector, which may help with cases such as firewalls preventing incoming connections from Graphistry to your database
- Run a bastion server between Graphistry and your database, such as a new Splunk search head
- Create fine-grained permissions by running multiple Graphistry virtual servers, with a new Splunk role per instance

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, edit `Caddyfile` (Caddy docs) and mount your certs by editing `docker-compose.yml` (Caddy Docker docs)

Sample free automatically-renewing LetsEncrypt TLS certificates with Caddy

Caddyfile

```
*.acme.org {
  tls {
    max_certs 100
  }
  proxy / nginx:80 {
    websocket
  }
}
:80 {
  proxy / nginx:80 {
    websocket
  }
}
```

Nginx (DEPRECATED)

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 17

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('m
```

=>

```
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 18

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 19

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&EIO
{...,"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 20

Planning a Graphistry Deployment

You can architect your deployment based on usage mode and environment constraints. We recommend starting with AWS/Azure Marketplace for most projects.

The following documents describe **common Graphistry configurations**, examples of **which configuration may make sense for you**, and **Graphistry's components**.

Related:

- **Capacity planning and host environment configuration**: See Recommended Deployment Configurations: Client, Server Software, Server Hardware
- **Graphistry server configuration**: See main configuration docs

1. Common Configurations

Graphistry deployments generally follow one of these patterns:

1. **Preconfigured AWS/Azure Marketplace instance** The most typical case.

- Preconfigured for one-click quicklaunch: Nvidia, Docker, Graphistry, Jupyter, connectors, and many DB/API samples
- Data stays in your team's private cloud account
- Great for POCs. Utility-fee pricing means cheap POCs by turning off the instance when inactive.

- Works well with databases also running securely in your cloud account and external APIs
- Dedicated Graphistry support

2. **Preconfigured AWS/Azure Marketplace instance with on-prem data bridging** On-prem enterprise data sources can likely be accessed as-is or via firewall changes. However, sometimes it is easier to use an on-prem jump box.

- Start with Graphistry AWS/Azure Marketplace as usual, e.g., for POCs
- Run the Graphistry Data Bridge closer to the DB/API, e.g., on-prem
- Configure the Graphistry Data Bridge, Graphistry server, and firewalls/DBs to work in bridged mode

3. **Manual GPU environment with Graphistry containers** Primarily useful for heavily regulated environments such as Federal and Financial teams with requirements such as air-gapping.

- Start with Graphistry AWS/Azure Marketplace as usual, e.g., for POCs
- When ready, follow Graphistry guides for Nvidia/Docker environment setup and Graphistry container installation
- Likely benefits from online Graphistry Support
- We recommend Cloud over On-Prem when possible

4. **Graphistry Cloud (Alpha)** Graphistry, Inc. runs and manages your Graphistry account, with a choice of shared and isolated tiers:

- Isolated tier: Useful when it is difficult to one-click launch on the Marketplace or procure/setup on-prem hardware
- Shared tier: Useful if you want the cost savings and expect limited use

2. Picking a Configuration

- **POC**: Save significant time by going with **AWS/Azure Marketplace**!
- Optional: Turn off Graphistry when not using it
- Do not use if you cannot put your data in the cloud
- Dedicated Graphistry staff is happy to assist with setup, configuration, demos, use cases, and other POC tasks
- If needed, Graphistry staff can help with Data Bridge setup
- **Analyst team**: Go with **AWS/Azure Marketplace** or **AWS/Azure BYOL**
- Do not use if you cannot put your data in the cloud

- Dedicated Graphistry staff is happy to assist with setup, configuration, demos, use cases, and other POC tasks
- If needed, Graphistry staff can help with Data Bridge setup
- **Analyst team, no AWS/Azure/GPU access**: Go with **Graphistry Cloud** (alpha). Please contact Graphistry for discussion of options and support.
- **Highly-regulated federal, bank, telco, utility, hospital**: Get started with **Graphistry Marketplace** for the POC, and go on-prem from there.
- **Commercial Product Developer**: To empower your product with Graphistry, we recommend POC'ing via Marketplace and using Graphistry Cloud's API (alpha) similar to how you might use Google Maps.

3. Graphistry Components

Understanding Graphistry's architecture may help you make your deployment architecture decisions:

- **Graphistry Core: The visualization server**
- GPU Visualization uploads and live sessions
- Docker containers that require nvidia-docker and Nvidial Pascal or later (See GPU/OS hardware requirements)
- Supports multi-GPU (single node) and GPU resource isolation
- Configurable reverse proxy, TLS, and user/API key management
- **Graphistry Investigation Server (Optional)**
- Installed as part of Graphistry Core
- Investigation templates
- Investigation cases, with embedded visualizations/sessions
- Connectors: Pushdown queries over DB/API connectors
- Deep linkable: External systems (Splunk, dashboards, emails, . . .) can be configured with links into Graphistry views and workflows
- **Graphistry Data Bridge (Optional)**
- Standalone docker-based server (See Graphistry Data Bridge)
- Enables individual Investigation Server connectors to proxy through a jump server
- Useful for bridging Graphistry cloud with on-prem data sources when simpler alternatives do not suffice
- **Graphistry API**
- Used by external web apps, notebook servers, etc.
- Communicates via language-neutral REST APIs and Graphistry-provided per-language convenience libraries
- **Data Science Notebooks**
- Jupyter is installed as part of Graphistry Core

- External notebooks may also be used via Graphistry API

Chapter 21

Chapter 22

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}
${resource}"
s3Key=xxxxxxxxxxxxxxxxxxx
s3Secret=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
signature=`/bin/echo -en "$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}
```

Chapter 23

Manual Graphistry Installation

We highly recommend getting started with quick launching a preconfigured AWS Marketplace or Azure Marketplace instance. When that is not an option, you may still be able to use a preconfigured GPU environment or automation script from Graphistry.

If none of those situations apply, read on for how to go to an unconfigured Linux instance to running Graphistry.

Contents

1. Prerequisites

- Graphistry-provided tarball

- Linux with Nvidia RAPIDS and Nvidia Docker via `Docker Compose 3`
- RAPIDS-compatible NVidia GPU: Pascal or later.

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

2. Instance Provisioning

AWS & Azure Marketplace

Skip almost all of these steps by instead running through AWS Marketplace and Azure Marketplace.

AWS & Azure BYOL

- **Start from an Nvidia instace** You can skip most of the steps by starting with an Nvidia NGC or Tensorflow instance.
- These still typically require installing `docker-compose`, setting `daemon.json` to default to the `nvidia-docker` runtime, and restarting `docker`.
- **Start from raw Ubuntu/RHEL** You can build from scratch by picking a fully unconfigured starting point and following the On-Prem instructions.

On-Prem

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

Airgapped

Graphistry runs airgapped without any additional configuration.

3. Linux Dependency Installation

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

See our sample scripts for RHEL 7.6 and Ubuntu 18.04 LTS. For automating this process, please contact Graphistry staff.

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!

```
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

Red Hat Enterprise Linux 7.6 (RHEL) manual configuration

We do *not* recommend manually installing the environment dependencies. Instead, use a Graphistry-managed Cloud Marketplace instance, a prebuilt cloud image, or another partner-supplied starting point.

However, sometimes a manual installation is necessary, or to otherwise run on RHEL instead of the provided Ubuntu AMIs.

The reference script below was last tested with an AWS RHEL 7.6 2019 AMI on a P3.2 (V100) and Nvidia RAPIDS 0.7.

- EPEL 7
- Nvidia driver 430.40
- Docker CE 19.03.1
- container-selinux-2.107-1.el7_6.noarch.rpm
- Docker-compose 1.24.1

Chapter 25

Manual RHEL environment configuration

Each subsection ends with a test command.

```
# Check HW
sudo yum -y install pciutils
lspci | grep -e VGA -ie NVIDIA

# EPEL
sudo yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
sudo yum upgrade kernel
sudo reboot now

# Nvidia Driver
#- https://www.nvidia.com/en-us/drivers/unix/
#=> Latest long-lived branch:
wget http://us.download.nvidia.com/XFree86/Linux-x86_64/430.40/NVIDIA-Linux-x86_64-430.40.ru
chmod +x ./NVIDIA-Linux-$(uname -m)-*.run

grep CONFIG_MODULE_SIG=y /boot/config-$(uname -r) && \
grep "CONFIG_MODULE_SIG_FORCE is not set" /boot/config-$(uname -r) && \
sudo ./NVIDIA-Linux-$(uname -m)-*.run -e || \
sudo ./NVIDIA-Linux-$(uname -m)-*.run

nvidia-smi

# Docker
sudo yum install -y yum-utils
sudo yum install -y http://mirror.centos.org/centos/7/extras/x86_64/Packages/container-selin
```

```
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install -y docker-ce
sudo systemctl enable --now docker

sudo docker run hello-world

# Docker-compose
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(ur
sudo chmod +x /usr/local/bin/docker-compose

docker-compose --version

# Nvidia docker runtime
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-container-runtime/$distribution/nvidia-container-
sudo yum install -y nvidia-container-runtime
sudo systemctl enable --now docker

sudo docker run --gpus all nvidia/cuda:9.0-base nvidia-smi

# Nvidia docker as default runtime (needed for docker-compose)
sudo yum install -y vim
sudo vim /etc/docker/daemon.json
{
    "default-runtime": "nvidia",
    "runtimes": {
        "nvidia": {
            "path": "/usr/bin/nvidia-container-runtime",
            "runtimeArgs": []
        }
    }
}
sudo systemctl restart docker

sudo docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi
sudo docker run --rm nvidia/cuda nvidia-smi
```

Chapter 26

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: +3s/min
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 27

Manual inspection of all key running components

Takes about 5-10min. See `Quick Testing` below for an expedited variant.

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

Login and get the API key from your dashboard homepage, or run the following:

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

- 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')`

6 Quick Testing

- `docker ps` reports no "unhealthy", "restarting", or prolonged "starting" services
- If a GPU service is unhealthy, the typical cause is an unhealthy Nvidia environment. Pinpoint the misconfiguration through the following progression:
- `docker run hello-world` reports a message <– tests Docker installation

- `nvidia-smi` reports available GPUs <– tests host drivers
- `docker run --gpus nvidia/cuda nvidia-smi` reports available GPUs <– tests nvidia-docker installation
- `docker run --runtime=nvidia nvidia/cuda nvidia-smi` reports available GPUs <– tests nvidia-docker installation
- `docker run --rm nvidia/cuda  nvidia-smi` reports available GPUs <– tests Docker defaults
- `docker run graphistry/cljs:1.1 npm test` reports success <– tests driver versioning
- "docker run –rm grph/streamgl-gpu:`cat VERSION`-dev nvidia-smi" reports available GPUs
- Pages load when logged in
- `site.com` shows Graphistry homepage and is stylized <– Static assets are functioning
- `site.com/graph/graph.html?dataset=Facebook` clusters and renders a graph
  - If the page loads but the graph is empty, see above instructions for testing Nvidia environment
  - Check browser console logs for WebGL errors
  - Check browser and network logs for Websocket errors, which may require a change in `Caddy` reverse proxying
- 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).
- Investigations
- `site.com/pivot` loads
- `site.com/pivot/connectors` loads a list of connectors
  - When clicking the `Status` button for each connector, it reports green. Check error reported in UI or docker logs (`docker compose logs -f -t pivot`): likely configuration issues such as password, URL domain vs fqdn, or firewall.
- Opening and running an investigation in `site.com/pivot` uploads and shows a graph

Chapter 28

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

Chapter 29

Update, Backup, and Migrate

Updates, backups, and migrations involve manipulating configurations, local data, the global postgres data volume, and Graphistry install itself.

Install multiple releases

We recommend the following folder structure:

```
graphistry/releases/v1.3.2/
graphistry/releases/v2.10.5/
...
```

If running concurrent versions of the same release:

Edit `docker-compose.yml`'s top "`volumes:`" section to use a unique prefix

```
volumes:
  ...
  postgres_data:
    name:  v2_10_5_MY_COPY_2_postgres_data
```

Launch under a unique name using `-p`:

```
docker-compose -p my_unique_name up -d
```

The config and data files

- **Local**: Configuration and certain data is kept at the root of the installed release
- `.env`, `Caddyfile`, `.caddy`, `etc/ssl`, `.pivot-db`
- Edits to `docker-compose.yml` (not recommeded)
- **Postgres**: Postgres volumes are managed by Docker
- `docker volume ls | grep postgres`
  - => `<version>_postgres_data`, `<version>_postgres_backups`
- `sudo ls -al /var/lib/docker/volumes | grep postgres`
  - => `<version>_postgres_data`, `<version>_postgres_backups`

Update

NOTE: Check release version history for any special instructions.

Graphistry maintains backwards-compatibility around data. New versions automatically handle migration issues around database and file format data conversions. Administrators are responsible for backing up and loading in the old data.

In practice, we recommend increasing service uptime nad minimizing administrator effort by doing installs on fresh cloud instances. However, you can generally reuse the box.

doing new installs on new instances, and instead m

- Put new release in a new folder
- Snapshsot the old instance, and ideally, while it is stopped
- `old_release $ docker-compose stop`
- Copy local config and data files into the new install folder
- Copy the Postgres data: see below
- Run the usual installation and launch procedure: `docker load -i containers.tar`, ...

Special case: REST API clients - PyGraphistry & JavaScript

REST API clients can often be directly from public repositories:

- PyGraphistry: `pip` and `github`
- React & JavaScript: `npm` and `github`

See central release notes for when new client API features require updating the server.

Special case: Postgres

Option A: Postgres-managed data migration:

The safest approach is to use `pgdump` to backup and `psql` to restore. See `.env` for `postgres` user/password/db configurations:

```
old_release $ docker-compose exec postgres  /usr/bin/pg_dump  -U graphistry graphistry | gzi

new_release $ docker-compose docker exec postgres psql -U graphistry graphistry < backup.sql
```

Option B: Direct file system copy:

If the underlying `postgres` container and your system environment have not changed, you may be able to simply copy the volumes:

```
cp -rp /var/lib/docker/volumes/<version_1>_postgres_data /var/lib/docker/volumes/<version_2>
cp -rp /var/lib/docker/volumes/<version_1>_postgres_backups /var/lib/docker/volumes/<version
```

See the Docker-managed instructions to do the same.

Backup & Migrate

Same as update.

Chapter 30

User Creation

Account Creation Model

- Graphistry temporarily starts with **Open Registration**
- The first user to register automatically gains an **admin** role, and the system automatically switches to **Invite-Only**
- **admin** users can create new users and assign them roles, including **admin**
- Every user gets an **API key**

Create Initial User: Admin 1

Step	Diagram
Upon starting Graphistry the first time, simply sign up. Take care to record your login/pwd.	

Create Subsequent Users & Roles

Step	Diagram
Go to the **Admin Portal**	
Open the **Users** manager	
Add user	
Set **Username**/**Password**	
Save and continue editing	
Set **Permissions**, typically unchecking **staff** and **superuser**	
Save	

Congrats, your user can now log in!

Provide API Keys

- Every user gets an API key on their dashboard page
- Additional API keys can be generated via CLI commands
- API keys are not currently revocable
- To preserve API keys across installations, make sure to copy `.env`