Contents

Managing a Graphistry Deployment

Quick Install

```
### Environment: Graphistry depends on nvidia-docker-2 and docker-compose
### Sample environment configuration for Ubuntu 16.04 cloud environments:
git clone https://github.com/graphistry/graphistry-cli.git
bash graphistry-cli/bootstrap.sh ubuntu-cuda9.2

### Install
docker load -i containers.tar
```

Commands

Start (interactive)	`docker-compose up`	Starts Graphistry, close with ctrl-c
Start (daemon)	`docker-compose up -d`	Starts Graphistry as background process
Stop	`docker-compose stop`	Stops Graphistry
Restart	`docker restart <CONTAINER>`	
Status	`docker-compose ps`, `docker ps`, and `docker status`	

API Key	docker-compose exec central curl -s http://localhost:10000/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`)	

Contents

Chapter 1

Instance & Environment Setup

1. Prerequisites

- Graphistry Docker container
- Linux with `nvidia-docker-2`, `docker-compose`, and `CUDA 9.2`. Ubuntu 16.04 cloud users can use a Graphistry provided environment bootstrapping script.
- NVidia GPU: K80 or later. Recommended G3+ on AWS and NC Series on Azure.
- Browser with Chrome or Firefox

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

2. Instance Provisioning

AWS

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

Proceed to the OS-specific instructions below.

For further information, see full AWS installation instructions.

Azure

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

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

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

Proceed to the OS-specific instructions below.

For further information, see full Azure installation instructions.

On-Premises

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

Airgapped

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

3. Linux Dependency Installation

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

Ubuntu 16.04 LTS

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

RHEL 7.4 / CentOS 7

Note: Temporarily not supported on AWS/Azure

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

After

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

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

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

Test environment

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

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

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

4. Graphistry Container Installation

`docker load -i containers.tar`

Congratulations, you have installed Graphistry!

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

Chapter 2

Configuration

Strongly Recommended:

After testing a base install works, configure the following:

- Setup `pivot` password
- Setup data persistence folders in case of restarts
- Generate API Key for developers & notebook users

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

Chapter 3

Maintenance

OS Restarts

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

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

Upgrading

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

Chapter 4

Testing:

Environment

If you downloaded the CLI:

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

Note that these are *not* deep tests of the environment.

Healthchecks

- Installation repositories are accessible:
- ping www.github.com
- ping shipyard.graphistry.com
- ping us.gcr.io
- Nvidia infrastructure setup correctly
- `nvidia-smi` reports available GPUs
- `nvidia-docker run nvidia/cuda nvidia-smi` reports available GPUs
- `nvidia-docker run graphistry/cljs:1.1 npm test` reports success (see airgapped alternative as well)
- Using the image listed in `docker images`, running `nvidia-docker run us.gcr.io/psychic-expanse-187412/graphistry/release/viz-app:1024 nvidia-smi` reports available GPUs
- Configurations were generated:
- `.config/graphistry/config.json`
- `httpd-config.json`
- `pivot-config.json`
- `viz-app-config.json`
- Services are running: `docker ps` reveals no restart loops on:
- `monolith-network-nginx`
- `monolith-network-pivot`
- `monolith-network-viz`
- `monolith-network-mongo`

- `monolith-network-db-bu`
- `monolith-network-pg`
- Services pass initial healthchecks:
- `site.com/central/healthcheck`
- `site.com/pivot/healthcheck`
- `site.com/worker/10000/healthcheck`
- Pages load
- `site.com` shows Graphistry homepage
- `site.com/graph/graph.html?dataset=Facebook` clusters and renders a graph
- `site.com/pivot` loads a list of investigations
- `site.com/pivot/connectors` loads a list of connectors
- ˆ When clicking the `Status` button for each connector, it reports green
- Opening and running an investigation in `site.com/pivot` uploads and shows a graph
- Data uploads
- Can generate an API key with the CLI: `graphistry` –> `keygen`
- Can use the key to upload a visualization: https://graphistry.github.io/docs/legacy/api/0.9.2/api.html#cur
- Can then open that visualization in a browser

Notebooks

Create the below notebook, fill in appropriate values for `GRAPHISTRY`. The expected result is a link, that when you click it, shows a graph with 3 nodes.

```
GRAPHISTRY = {
    'server': 'my.server.com', #no http, just domain
    'protocol': 'http',
    'key':  'MY_API_KEY'
}

!pip install pandas
import pandas as pd
edges_df=pd.DataFrame({'src': [0,1,2], 'dest': [1,2,0]})

!pip install graphistry
import graphistry
graphistry.register(**GRAPHISTRY)

graphistry.bind(source='src', destination='dest').edges(edges_df).plot(render=False)
```

For further information about the Notebook client, see the OSS project PyGraphistry (PyPI).

Chapter 5

Troubleshooting

See further documentation.

Chapter 6

Configuring Graphistry

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

Four configurations: .env, docker-compose.yml, pivot.json, and etc/ssl/*

- Graphistry is configured through a `.env` file, which is what you primarily edit
- It can be used to enable a `data/pivot.json`, which supports the same commands, but is more convenient for heavier configurations such as json ontologies
- The `docker-compose.yml` reads the `.env` file, and more advanced administrators may edit the yml file as well. Maintenance is easier if you never edit it.
- TLS is via `etc/ssl/*`

Backup your configuration

Graphistry tarballs contain default `.env` and `docker-compose.yml`, so make sure you put them in safe places.

If you create `json` config files, such as a `data/config/pivot.json`, back them up too.

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

Persist user data across restarts

We recommend persisting data to `${PWD}/data/config/{pivot.json,viz.json}` and `${PWD}/data/investigatigations,viz}`, or the same on a network mount, and running regular backups.

Configure your `.env` and `docker-compose.yml` as follows:

data/config/{pivot,viz}.json

Create `data/config/pivot.json` and `data/config/viz.json` with the value `{}`:

```
mkdir -p data/config && echo "{}" > data/config/pivot.json &&
echo "{}" > data/config/viz.json
```

.env

Eanble in `.env`:

```
GRAPHISTRY_CONFIG=./data/config
GRAPHISTRY_INVESTIGATIONS=./data/investigations
GRAPHISTRY_VIZ=./data/viz
PIVOT_CONFIG_FILES=/opt/graphistry/config/pivot.json
VIZ_CONFIG_FILES=/opt/graphistry/config/viz.json
GRAPHISTRY_INVESTIGATIONS_CONTAINER_DIR=/opt/graphistry/apps/core/pivot/data
GRAPHISTRY_VIZ_CONTAINER_DIR=/tmp/graphistry
```

These will enable reading of `data/config/{pivot,viz}.json` and write user data to `data/{investigations,viz}`.

docker-compose.yml

Enable in `docker-compose.yml` if not already there:

```
viz:
    ...
    volumes:
      - ${GRAPHISTRY_VIZ}:${GRAPHISTRY_VIZ_CONTAINER_DIR}
```

and

```
pivot:
    ...
    volumes:
       - ${GRAPHISTRY_CONFIG}:/opt/graphistry/config
       - ${GRAPHISTRY_INVESTIGATIONS}:${GRAPHISTRY_INVESTIGATIONS_CONTAINER_DIR}
```

Connectors

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

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

Finer-grained configurations are easier to setup and maintain via `data/config/pivot.json`.

Your Graphistry support engineer can provide examples.

Passwords

Graphistry passwords are random across container runs so you likely want to override and keep.

Uncomment and edit lines of `.env` for `PIVOT_PASSWORD_HASH` (see instructions in `.env`) for the password used on `your-site.com/pivot`

Your API keys should be stable across runs. If not, contact your Graphistry support engineer.

Ontology

Setup Graphistry for data persistence (see above), and then in `data/config/pivot.json`, add and configure the below.

- 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"
        }
    }
}
```

Nginx Config and SSL

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:
    image: spengler.grph.xyz/release/nginx-proxy:2000
    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 7

Recommended Deployment Configurations: Client, Server Software, Server Hardware

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

Contents

Overview

- **Client**: Chrome/Firefox from the last 3 years, WebGL enabled, and 100KB/s download ability
- **Server**:
- x86 Linux server with 4+ CPU cores, 16+ GB CPU RAM (3GB per concurrent user), and 1+ Nvidia GPUs (K80 onwards) with 4+ GB RAM each (1+ GB per concurrent user)
- Recommend Ubuntu 16.04, 4+ CPU cores, 64GB+ CPU RAM, Nvidia Tesla or later

- Docker / CUDA 9.2 / nvidia-docker-2

Client

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

The Graphistry client runs in standard browser configurations:

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

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

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

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

Cloud

Tested AWS Instances:

- P2
- G3 ***Recommended for testing and initial workloads***
- P3

Tested Azure Instances:

- NV6 ***Recommended for testing and initial workloads***
- NC6

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

OS & Docker

Graphistry regularly runs on:

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

Both support nvidia-docker-2:

- Docker
- nvidia-docker-2
- CUDA 9.2

For cloud users, we maintain bootstrap scripts, and they are a useful reference for on-premises users.

User: Root vs. Not, Permissions

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

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

Storage

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

Server: Hardware Capacity Planning

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

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

Network

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

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

GPUs & GPU RAM

The following Nvidia GPUs are known to work with Graphistry:

- Tesla: K40, K80, M40
- DGX: P100, V100 ***Recommended***

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

CPU Cores & CPU RAM

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

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

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

Graphistry 1.0 virtualizes a single GPU for shared use by multiple users.

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

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

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

Chapter 8

Graphistry on AWS: Environment Setup Instructions

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

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

Contents:

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

Subsequent reading: General installation

Chapter 9

1. Pick Linux distribution

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

Ubuntu 16.04 LTS

- Available on official AWS launch homepage
- Find AMI for region https://cloud-images.ubuntu.com/locator/
- Ex: Amazon AWS us-east-1 xenial 16.04 amd64 hvm-ssd 20180405 ami-6dfe5010
- Follow provisioning instructions from AWS install
- G3 or P2: 200 GB, add a name tag, ssh/http/https; use & store an AWS keypair
- Login: ssh -i . . . private_key.pem ubuntu@public.dns

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

Chapter 10

2. Configure instance

- Instance: g3+ or p*
- 200GB+ RAM
- Security groups: ssh, http, https

Chapter 11

3. General installation

Proceed to the instructions for general installation.

Chapter 12

Graphistry on Azure: Environment Setup Instructions

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

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

Contents:

Subsequent reading: General installation

Prerequisites: Azure GPU Quota

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

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

Testing if you already have GPU quota

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

Requesting Azure for GPU Quota

For each location in which you want to run Graphistry:

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

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

1. Start a new GPU virtual machine

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

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

2. Confirm proper instance

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

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

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

3. Proceed to general Graphistry installation

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

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

Chapter 13

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.

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`
  - See https://www.npmjs.com/package/@graphistry/cljs

  - *note*: Requires CL installed in host, which Graphistry normally 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
- 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 14

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 le
{...,"msg":"constructor:  function Uint8Array() { [native code] }","time":"2018-07-20T06:56:
{...,"counts":{"num":176468,"offset":0},"msg":"Copying hostBuffer[logicalEdges]. Orig Buffe
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:5
{...,"counts":{"num":88234,"offset":0},"msg":"Copying hostBuffer[forwardsEdgeToUnsortedEdge
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:5
{...,"counts":{"num":176468,"offset":0},"msg":"Copying hostBuffer[edgeColors]. Orig Buffer
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:5
{...,"counts":{"num":4039,"offset":0},"msg":"Copying hostBuffer[pointColors]. Orig Buffer le
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:5
{...,"counts":{"num":8078,"offset":0},"msg":"Copying hostBuffer[forwardsEdgeStartEndIdxs].
{...,"msg":"constructor:  function Uint32Array() { [native code] }","time":"2018-07-20T06:5
{...,"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_97olrAAA
{...,"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_97olrAAA
{...,"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_97olrAAA
{...,"msg":"HTTP GET request for vbo forwardsEdgeToUnsortedEdge","time":"2018-07-20T06:56:1
{...,"msg":"HTTP request received by Express.js { originalUrl: '/vbo?id=9ckImeuIxO_97olrAAA
{...,"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_97olrAAA
{...,"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_97olrAAA
{...,"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 15

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 loca
```

=> 10+ workers

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

Browser

A. Can access site:

Browse to

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

=>

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

B. Browser has web sockets enabled

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

C. Can follow central redirect:

Open browser developer network analysis panel and visit

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

=>

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

This call sequence stress a lot of the pipeline.

NGINX

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

A. Can server central routes

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

=>

```
HTTP/1.1 200 OK
```

B. Can receive central redirect:

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

=>

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

and

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

=>

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

C. Can serve worker routes

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

=>

```
HTTP/1.1 200 OK
```

Viz container

A. Container has a running central server

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

=>

```
HTTP/1.1 200 OK
```

and

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

=>

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

C. Can communicate with Mongo

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

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

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

=>

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

D. Has running workers

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

=>

```
HTTP/1.1 200 OK
```