Deployment: Monitoring

CS 485/698: AI-Assisted SE

Today's Agenda

- Team meeting (~10 minutes)
  - Standup: is your project deployable yet? If not, what are the blockers that you need to deal with so that it is?
  - If you've already deployed your project: how do you know that it is working *right now*?
- Lecture on monitoring
- In-class activity: red team exercise on project deployment pipelines

Monitoring

- **Today's key question**: once we've deployed an application, how do we know that it's **still working**?

Monitoring

- **Today's key question**: once we've deployed an application, how do we know that it's **still working**?
- What kinds of things can go wrong with a running application?

Monitoring

- **Today's key question**: once we've deployed an application, how do we know that it's **still working**?
- What kinds of things can go wrong with a running application?
  - Availability**:** did the service go down?
  - Authentication: can users log in?
  - Authorization: do users have appropriate permissions?
  - Data Privacy: did user data get leaked?
  - Security: did a crypto key get leaked or expire? Is there some kind of cross-site scripting or DDOS attack occurring?
  - Etc.

Monitoring

- **Today's key question**: once we've deployed an application, how do we know that it's **still working**?
- What kinds of things can go
  - Availability: did the ser
  - Authentication: can use
  - Authorization: do users
  - Data Privacy: did user d
  - Security: did a crypto k
    kind of cross-site scriptin
  - Etc.

Monitoring is the automated systems that collect metrics about your running system

Monitoring

- **Today's key question**: once we've deployed an application, how do we know that it's **still working**?
- What kinds of things can go
  - Availability: did the ser
  - Authentication: can use
  - Authorization: do users
  - Data Privacy: did user d
  - Security: did a crypto k
    kind of cross-site scriptin
  - Etc.

Monitoring is the automated systems that collect metrics about your running system

- without monitoring, you have no way to tell whether the service is **even working**

Monitoring: Logging

- Probably the first way to “check if your program is working” that you learned:

Monitoring: Logging

- Probably the first way to “check if your program is working” that you learned:
  - emit strings from your code to the terminal
    - e.g., `printf(“got to here.”)`
  - sometimes called ***printf debugging***

Monitoring: Logging

- Probably the first way to “check if your program is working” that you learned:
  - emit strings from your code to the terminal
    - e.g., `printf(“got to here.”)`
  - sometimes called ***printf debugging***
    - key idea: **instrument** the program so that it prints the values of key variables at a particular point

Monitoring: Logging

- Probably the first way to “check if your program is working” that you learned:
  - emit strings from your code to the terminal
    - e.g., `printf(“got to here.”)`
  - sometimes called ***printf debugging***
    - key idea: **instrument** the program so that it prints the values of key variables at a particular point
  - but really this is a primitive form of one of the most important monitoring techniques for modern software applications: ***logging***

Monitoring: Logging

- Probably the first way to "check if your program is working" that you learned:
  - emit strings from your code to the terminal
    - e.g., `printf("got to here.")`
  - sometimes called ***printf debugging***
    - key idea: **instrument** values of key variab
  - but really this is a primit monitoring techniques ***logging***

Definition: ***logging*** is the process of recording information about the program's internal state as it runs via a printf-like interface

Logging: levels

Typical example of a (Java) logging statement:

```
log.debug("myVariable=%s", myVariable);
```

Logging: levels

Typical example of a (Java) logging statement:

```
log.debug("myVariable=%s", myVariable);
```

the log itself is usually a static field; the logging framework instantiates it, etc.

Logging: levels

Typical example of a (Java) logging statement:

```
log.debug("myVariable=%s", myVariable);
```

"debug" means if debug-level logging isn't enabled in the framework, this becomes a no-op

Logging: levels

Typical example of a (Java) logging statement:

```
log.debug("myVariable=%s", myVariable);
```

"debug" means if debug-level logging isn't enabled in the framework, this becomes a no-op

levels:
error ⊆ warning ⊆ info ⊆ debug

developer chooses one level, all lower level messages are also logged

Logging: levels

Typical example of a (Java) logging statement:

```
log.debug("myVariable=%s", myVariable);
```

printf-like syntax isn't just for show: goal here is **lazy evaluation**, so that if debug logging isn't enabled, this string is never constructed

Logging: levels

Typical example of a (Java) logging statement:

```
log.debug("myVariable=%s", myVariable);
```

arguments to printf passed by reference, so if debug-level logging is off, this argument's toString() method is never called

Logging: what to log

- Consider both ***direct*** (e.g. business) metrics, and ***indirect*** (e.g. system) metrics

Logging: what to log

- Consider both ***direct*** (e.g. business) metrics, and ***indirect*** (e.g. system) metrics
  - **Application level**: transactions completed, click-through or conversion rates, status of 3rd party components

Logging: what to log

- Consider both ***direct*** (e.g. business) metrics, and ***indirect*** (e.g. system) metrics
  - **Application level**: transactions completed, click-through or conversion rates, status of 3rd party components
  - **Middleware level**: memory, thread/db connection pools, connections, response time

Logging: what to log

- Consider both ***direct*** (e.g. business) metrics, and ***indirect*** (e.g. system) metrics
  - **Application level**: transactions completed, click-through or conversion rates, status of 3rd party components
  - **Middleware level**: memory, thread/db connection pools, connections, response time
  - **OS level**: memory usage, swap usage, disk space, CPU load

Logging: what to log

- Consider both ***direct*** (e.g. business) metrics, and ***indirect*** (e.g. system) metrics
  - **Application level**: transactions completed, click-through or conversion rates, status of 3rd party components
  - **Middleware level**: memory, thread/db connection pools, connections, response time
  - **OS level**: memory usage, swap usage, disk space, CPU load
  - **Hardware level**: voltages, temperatures, fan speeds, component health

Logging: advice

- **Do** log lots of information at debug or info level, so that if something is wrong with your service you can quickly get lots of information that you can use to debug it.

Logging: advice

- **Do** log lots of information at debug or info level, so that if something is wrong with your service you can quickly get lots of information that you can use to debug it.
- **Don't** log sensitive data (e.g., credit card numbers in plaintext!)
  - this is a surprisingly common and important problem - developers have a tendency to log anything that might be useful when debugging a failure later!

Metrics and dashboards

- We don’t want to have to read all of our logs, all of the time

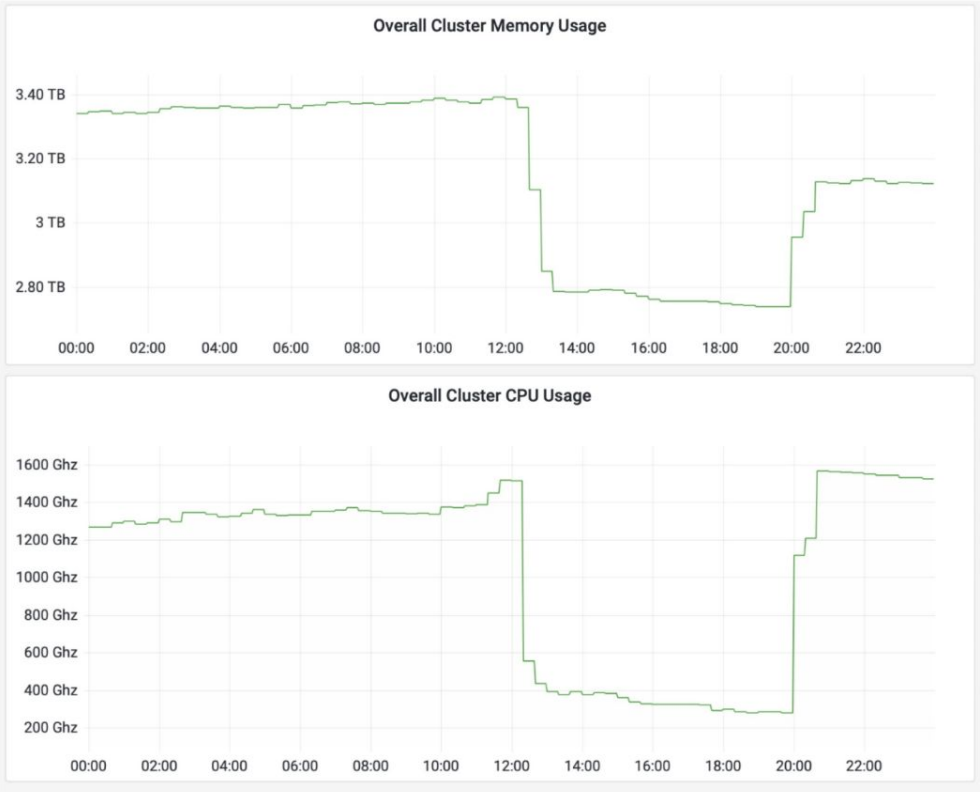
Metrics and dashboards

- We don’t want to have to read all of our logs, all of the time
  - Instead, we want to define quantitative ***metrics*** that are computed from our logs that indicate service health

Metrics and dashboards

- We don’t want to have to read all of our logs, all of the time
  - Instead, we want to define quantitative ***metrics*** that are computed from our logs that indicate service health
  - Typically, these are visualized using ***dashboards***

Metrics and dashboards: example dashboards



Grafana (AGPL, c 2014)

InfluxDB (MIT license, c 2013)

Metrics and dashboards

- We don’t want to have to read all of our logs, all of the time
  - Instead, we want to define quantitative ***metrics*** that are computed from our logs that indicate service health
  - Typically, these are visualized using ***dashboards***
- Your monitoring system should automatically ***alert*** you whenever a metric is doing something unexpected

Metrics and dashboards

- We don’t want to have to read all of our logs, all of the time
  - Instead, we want to define quantitative ***metrics*** that are computed from our logs that indicate service health
  - Typically, these are visualized using ***dashboards***
- Your monitoring system should automatically ***alert*** you whenever a metric is doing something unexpected
  - This might be an email or a ticket (if it’s not urgent)

Metrics and dashboards

- We don’t want to have to read all of our logs, all of the time
  - Instead, we want to define quantitative ***metrics*** that are computed from our logs that indicate service health
  - Typically, these are visualized using ***dashboards***
- Your monitoring system should automatically ***alert*** you whenever a metric is doing something unexpected
  - This might be an email or a ticket (if it’s not urgent)
  - Or it might be a ***page*** to the on-call engineer, if the problem needs to be investigated right away

Subtleties in metrics

Subtleties in metrics

- For simplicity and usability, we often aggregate raw measurements. This needs to be done **carefully**.

Subtleties in metrics

- For simplicity and usability, we often aggregate raw measurements. This needs to be done **carefully**.
- e.g., consider “the number of requests per second served”

Subtleties in metrics

- For simplicity and usability, we often aggregate raw measurements. This needs to be done **carefully**.
- e.g., consider “the number of requests per second served”
  - even this apparently straightforward measurement **implicitly aggregates** data over the measurement window

Subtleties in metrics

- For simplicity and usability, we often aggregate raw measurements. This needs to be done **carefully**.
- e.g., consider “the number of requests per second served”
  - even this apparently straightforward measurement **implicitly aggregates** data over the measurement window
- We need to consider questions like “Is the measurement obtained once a second, or by averaging requests over a minute?”

Subtleties in metrics

- For simplicity and usability, we often aggregate raw measurements. This needs to be done **carefully**.
- e.g., consider “the number of requests per second served”
  - even this apparently straightforward measurement **implicitly aggregates** data over the measurement window
- We need to consider questions like “Is the measurement obtained once a second, or by averaging requests over a minute?”
  - The latter may **hide** much higher instantaneous request rates in bursts that last for only a few seconds

Subtleties in metrics

- For simplicity and usability, y
  measurements. This needs t
- e.g., consider “the number o
  - even this apparently stra
    aggregates data over th
- We need to consider questi
  once a second, or by averagi
  - The latter may **hide** much higher instantaneous request rates in bursts that last for only a few seconds

E.g., consider two systems:

- system A serves 200 requests in every even-numbered second, and 0 requests in every odd-numbered second
- system B serves 100 requests every second

Subtleties in metrics

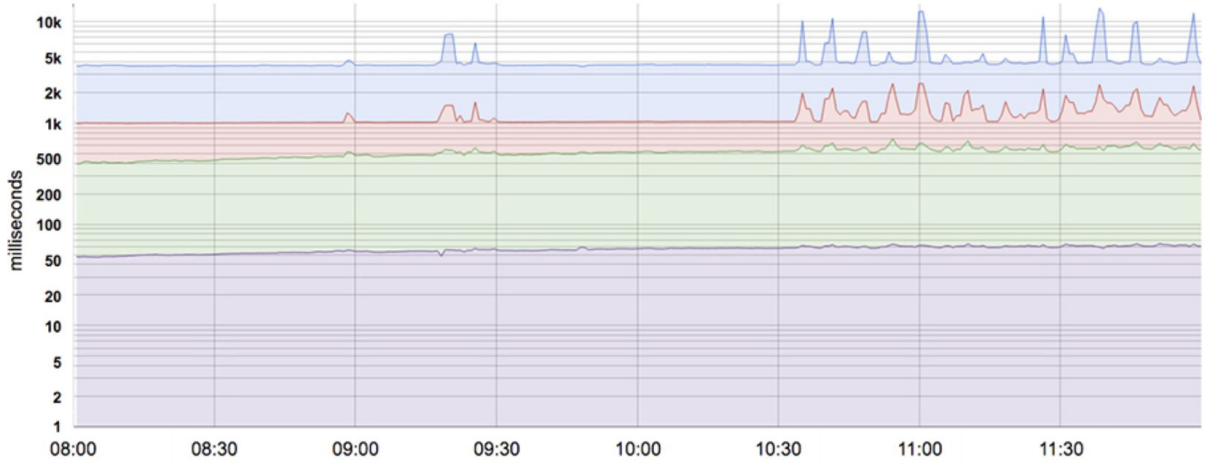
- It is better to view metrics as **distributions** (as in statistics) rather than as averages
  - this avoids hiding details like the example on the last slide

Subtleties in metrics

- It is better to view metrics as **distributions** (as in statistics) rather than as averages
  - this avoids hiding details like the example on the last slide

Subtleties in metrics

- It is better to view metrics as **distributions** (as in statistics) rather than as averages
  - this avoids hiding details like the example on the last slide



purple is 50th % latency

Subtleties in metrics

- It is better to view metrics as **distributions** (as in statistics) rather than as averages
  - this avoids hiding details like the example on the last slide

green is 85th % latency

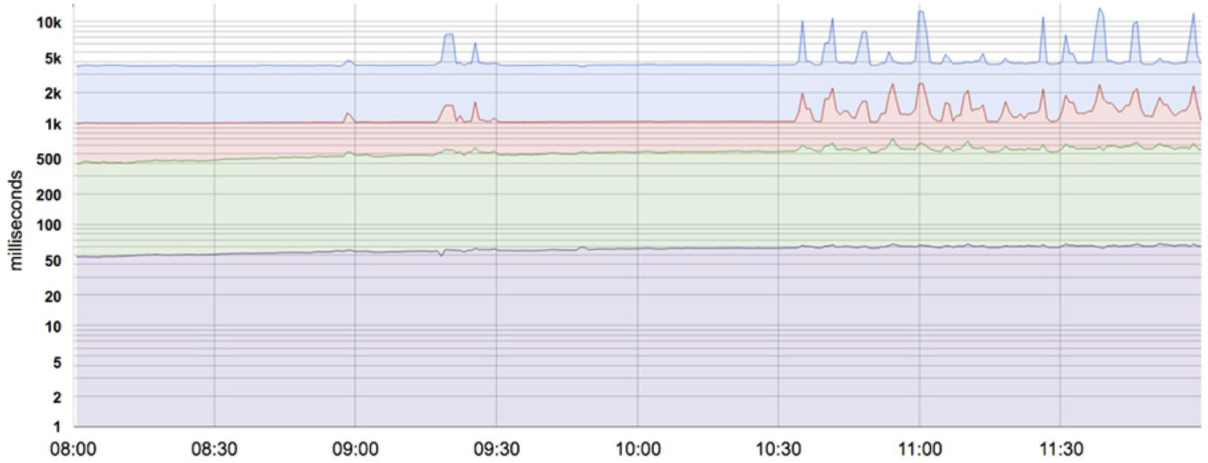
Subtleties in metrics

- It is better to view metrics as **distributions** (as in statistics) rather than as averages
  - this avoids hiding details like the example on the last slide

red is 95th % latency

Subtleties in metrics

- It is better to view metrics as **distributions** (as in statistics) rather than as averages
  - this avoids hiding details like the example on the last slide



blue is 99th % latency

Advice: choosing metrics

Advice: choosing metrics

- don’t pick target metrics based on **current system performance**
  - this just enshrines the status quo; instead, focus on user needs

Advice: choosing metrics

- don’t pick target metrics based on **current system performance**
  - this just enshrines the status quo; instead, focus on user needs
- keep it **simple** and avoid absolutes
  - e.g., don’t promise “infinite scaling” or “100% availability”

Advice: choosing metrics

- don't pick target metrics based on **current system performance**
  - this just enshrines the status quo; instead, focus on user needs
- keep it **simple** and avoid absolutes
  - e.g., don't promise "infinite scaling" or "100% availability"
- watch out for ***Goodhart's Law***: *"When a measure becomes a target, it ceases to be a good measure."*
  - regularly re-examine your chosen metrics to make sure they actually correlate with qualitative "good" system health

Advice: choosing metrics

- don't pick target metrics based on **current system performance**
  - this just enshrines the status quo; instead, focus on user needs
- keep it **simple** and avoid absolutes
  - e.g., don't promise "infinite scaling" or "100% availability"
- watch out for ***Goodhart's Law***: *"When a measure becomes a target, it ceases to be a good measure."*
  - regularly re-examine your chosen metrics to make sure they actually correlate with qualitative "good" system health
- avoid the ***McNamara Fallacy***: measure whatever can be easily measured and disregard that which cannot be measured easily

In-class Activity: Deployment Red Teams

- Each project team should have one **blue letter**
  - If your team has a “deployment expert”, give them the blue letter (swapping if necessary)
  - If you’re on a team by yourself, make sure your letter is blue
- If you have a **red letter**, you should find the person with the corresponding blue letter (note there are duplicate red letters)

In-class Activity: Deployment Red Teams

- **Goal**: the red team member(s) will audit the blue team's deployment pipeline
  - **Blue team**: start by explaining how your app is deployed to your red team
  - **Red team**: think about whether the blue team could improve…
    - deployment automation?
    - quality assurance/testing gates/CI
    - staged deploys?
    - monitoring infrastructure?
  - **Red team**: can you "sneak" a bad change through this pipeline?

In-class Activity: Deployment Red Teams

- **Goal**: the red team member(s)
  deployment pipeline
  - **Blue team**: start by expl
    your red team
  - **Red team**: think about w…
    - deployment automa
    - quality assurance/te
    - staged deploys?
    - monitoring infrastructure?
  - **Red team**: can you "sneak" a bad change through this pipeline?

By end of class, red team should open at least one PR that:

- documents a weakness that you discovered today
- or, even better, fixes a weakness

For participation credit. Use git's co-authoring feature to credit your whole team.

In-class Activity: Deployment Red Teams

- **Goal**: the red team member(s) will audit the blue team's deployment pipeline
  - **Blue team**: start by explaining how your app is deployed to your red team
  - **Red team**: think about whether the blue team could improve…
    - deployment automation?
    - quality assurance/testing gates/CI
    - staged deploys?
    - monitoring infrastructure?
  - **Red team**: can you "sneak" a bad change through this pipeline?

Wrapup and Reminders

- A6 due Wednesday (discussion in class)
- P6 due Sunday night
- Do your course evaluations!
  - I will take the feedback very seriously
  - For those that took 490: what should I include from this class in future version of 490?
- My office hours this week are cancelled
  - Make an appointment if you want to talk to me