DevOps (1/2)

Martin Kellogg

DevOps (1/2)

Today's agenda:

- **Reading Quiz**
- Operations, Toil, and the DevOps philosophy
- Ops challenge: deployment
- Achieving reliability
  - the service reliability hierarchy + SLAs/targets
  - monitoring
  - incident/emergency response
  - post-mortems + learning from failure

Reading quiz: DevOps (1/2)

Q1: The key difference in the composition between Google SRE teams and “traditional” operations teams is:

A. Google SRE teams are at least 50% software engineers
B. Google’s hiring bar is higher than traditional operations teams’
C. Google only hires people with CS degrees

Q2: Which is **not** recommended to prevent future emergencies?

A. Keeping a history of past outages
B. Adding manual double-checking to anything that’s failed before
C. Asking big, even improbable, questions about what might fail

Reading quiz: DevOps (1/2)

Q1: The key difference in the composition between Google SRE teams and “traditional” operations teams is:

A. Google SRE teams are at least 50% software engineers
B. Google’s hiring bar is higher than traditional operations teams’
C. Google only hires people with CS degrees

Q2: Which is **not** recommended to prevent future emergencies?

A. Keeping a history of past outages
B. Adding manual double-checking to anything that’s failed before
C. Asking big, even improbable, questions about what might fail

Reading quiz: DevOps (1/2)

Q1: The key difference in the composition between Google SRE teams and “traditional” operations teams is:

A. Google SRE teams are at least 50% software engineers
B. Google’s hiring bar is higher than traditional operations teams’
C. Google only hires people with CS degrees

Q2: Which is **not** recommended to prevent future emergencies?

A. Keeping a history of past outages
B. Adding manual double-checking to anything that’s failed before
C. Asking big, even improbable, questions about what might fail

DevOps (1/2)

Today's agenda:

- Reading Quiz
- **Operations, Toil, and the DevOps philosophy**
- Ops challenge: deployment
- Achieving reliability
  - the service reliability hierarchy + SLAs/targets
  - monitoring
  - incident/emergency response
  - post-mortems + learning from failure

DevOps (1/2)

Today's agenda:

- Reading Quiz
- **Operations, Toil, and**
- Ops challenge: deplo
- Achieving reliability
  - the service reliab
  - monitoring
  - incident/emergency response
  - post-mortems + learning from failure

Announcements:

- my office hours next week are on **Tuesday** (not Monday), 2-3pm
- sprint 3 is ending tomorrow (schedule your retro meetings!)

Operations

Definition: ***operations*** refers to anything that happens after the developers (think that they) are done building the software, including:

Operations

Definition: ***operations*** refers to anything that happens after the developers (think that they) are done building the software, including:

- setting up the servers that will run the software and installing the software on them

Operations

Definition: ***operations*** refers to anything that happens after the developers (think that they) are done building the software, including:

- setting up the servers that will run the software and installing the software on them
- conducting system/acceptance tests

Operations

Definition: ***operations*** refers to anything that happens after the developers (think that they) are done building the software, including:

- setting up the servers that will run the software and installing the software on them
- conducting system/acceptance tests
- running the software and keeping it running

Operations

Definition: ***operations*** refers to anything that happens after the developers (think that they) are done building the software, including:

- setting up the servers that will run the software and installing the software on them
- conducting system/acceptance tests
- running the software and keeping it running
- measuring the performance of the running software

Operations

Definition: ***operations*** refers to anything that happens after the developers (think that they) are done building the software, including:

- setting up the servers that will run the software and installing the software on them
- conducting system/acceptance tests
- running the software and keeping it running
- measuring the performance of the running software
- fixing any problems that arise while the software is running

Operations

Definition: ***operations*** refers to anything that happens after the developers (think that they) are done building the software, including:

- setting up the servers that will run the software and installing the software on them
- conducting system/acceptance tests
- running the software and keeping it running
- measuring the performance of the running software
- fixing any problems that arise while the software is running
- deploying new versions of the software

Operations: the traditional approach

Operations: the traditional approach

- traditionally, operations are mostly conducted by **system administrators** (or **sysadmins**) rather than by developers

Operations: the traditional approach

- traditionally, operations are mostly conducted by **system administrators** (or **sysadmins**) rather than by developers
  - sysadmins are specialists in specific tech stacks
    - e.g., experts at Linux or Windows, etc.

Operations: the traditional approach

- traditionally, operations are mostly conducted by **system administrators** (or **sysadmins**) rather than by developers
  - sysadmins are specialists in specific tech stacks
    - e.g., experts at Linux or Windows, etc.
  - e.g., NJIT's IT degree program was (probably) originally intended as preparation for this kind of role

Operations: the traditional approach

- traditionally, operations are mostly conducted by **system administrators** (or **sysadmins**) rather than by developers
  - sysadmins are specialists in specific tech stacks
    - e.g., experts at Linux or Windows, etc.
  - e.g., NJIT's IT degree program was (probably) originally intended as preparation for this kind of role
- this approach is best when systems **change rarely**

Operations: the traditional approach

- traditionally, operations are mostly conducted by **system administrators** (or **sysadmins**) rather than by developers
  - sysadmins are specialists in specific tech stacks
    - e.g., experts at Linux or Windows, etc.
  - e.g., NJIT's IT degree program was (probably) originally intended as preparation for this kind of role
- this approach is best when systems **change rarely**
  - e.g., when software is released on physical media

Operations: the traditional approach

- traditionally, operations are mostly conducted by **system administrators** (or **sysadmins**) rather than by developers
  - sysadmins are specialists in specific tech stacks
    - e.g., experts at Linux or Windows, etc.
  - e.g., NJIT's IT degree program was (probably) originally intended as preparation for this kind of role
- this approach is best when systems **change rarely**
  - e.g., when software is released on physical media
  - other advantages: easy to staff for, off-the-shelf tooling, etc.

Traditional ops in different business models

- two business models:

Traditional ops in different business models

- two business models:
  - **services** (i.e., the developing organization runs the software and sells access to customers)

Traditional ops in different business models

- two business models:
  - **services** (i.e., the developing organization runs the software and sells access to customers)
    - service ops: need to set up the servers/machines on which the software will run, install the software + dependencies, configure firewalls, etc.

Traditional ops in different business models

- two business models:
  - **services** (i.e., the developing organization runs the software and sells access to customers)
    - service ops: need to set up the servers/machines on which the software will run, install the software + dependencies, configure firewalls, etc.
  - **products** (i.e., sell/lease the software to others to run)

Traditional ops in different business models

- two business models:
  - **services** (i.e., the developing organization runs the software and sells access to customers)
    - service ops: need to set up the servers/machines on which the software will run, install the software + dependencies, configure firewalls, etc.
  - **products** (i.e., sell/lease the software to others to run)
    - product ops: still need to system test in the anticipated operating environment(s), set up servers providing those environments, install the software + dependencies, etc.

Traditional ops in di

Traditional approach to operations can work in either of these models!

- two business models:
  - **services** (i.e., the developing organization runs the software and sells access to customers)
    - service ops: need to set up the servers/machines on which the software will run, install the software + dependencies, configure firewalls, etc.
  - **products** (i.e., sell/lease the software to others to run)
    - product ops: still need to system test in the anticipated operating environment(s), set up servers providing those environments, install the software + dependencies, etc.

Operations: the traditional approach

- However, the traditional sysadmin approach to operations has downsides, too:

Operations: the traditional approach

- However, the traditional sysadmin approach to operations has downsides, too:
  - for services, ops costs **scale with system load**: more users = must hire more sysadmins to administer more servers, etc.

Operations: the traditional approach

- However, the traditional sysadmin approach to operations has downsides, too:
  - for services, ops costs **scale with system load**: more users = must hire more sysadmins to administer more servers, etc.
  - separation of operations and development means developers are not **directly exposed** to the costs of poor design decisions
    - this is a misalignment of incentives

Operations: the traditional approach

- However, the traditional sysadmin approach to operations has downsides, too:
  - for services, ops costs **scale with system load**: more users = must hire more sysadmins to administer more servers, etc.
  - separation of operations and development means developers are not **directly exposed** to the costs of poor design decisions
    - this is a misalignment of incentives
  - developers and sysadmins have different backgrounds, terminology, etc., leading to **communication breakdowns**

Operations: the traditional approach

- However, the traditional
  downsides, too:
  - for services, ops c
    must hire more sy
  - separation of ope
    are not **directly e**
    - this is a misali
  - developers and sy
    terminology, etc., leading to **communication breakdowns**

These problems **do not** mean that the traditional approach to operations is bad in all circumstances!

Operations: the traditional approach

- However, the traditional
  downsides, too:
  - for services, ops c
    must hire more sy
  - separation of ope
    are not **directly e**
    - this is a misali
  - developers and sy
    terminology, etc., leading to **communication breakdowns**

These problems **do not** mean that the traditional approach to operations is bad in all circumstances!

- But, they are serious concerns for modern systems with high release cadences, especially those that are:
  - microservices
  - delivered via the web
  - use “continuous delivery”

Operations: the DevOps approach

Operations: the DevOps approach

Key idea: combine the development and operations teams

Operations: the DevOps approach

Key idea: combine the development and operations teams

- “DevOps” is a portmanteau of “developers” + “operators”

Operations: the DevOps approach

Key idea: combine the development and operations teams

- “DevOps” is a portmanteau of “developers” + “operators”
- DevOps teams are organized around services/projects
    - what does this sound similar to?

Operations: the DevOps approach

Key idea: combine the development and operations teams

- “DevOps” is a portmanteau of “developers” + “operators”
- DevOps teams are organized around services/projects
  - what does this sound similar to?
- operational burden is **shared** by the developers who are building the system

Operations: the DevOps approach

Key idea: combine the development and operations teams

- “DevOps” is a portmanteau of “developers” + “operators”
- DevOps teams are organized around services/projects
  - what does this sound similar to?
- operational burden is **shared** by the developers who are building the system
  - better alignment of incentives between developers and operators, since same people perform both roles

Operations: the DevOps approach

Key idea: combine the development and operations teams

- “DevOps” is a portmanteau of “developers” + “operators”
- DevOps teams are organized around services/projects
  - what does this sound similar to?
- operational burden is **shared** by the developers who are building the system
  - better alignment of incentives between developers and operators, since same people perform both roles
- encourage operators to automate **toil**

Operations: the DevOps approach

Key idea: combine the development and operations teams

- “DevOps” is a portmanteau of “developers” + “operators”
- DevOps teams are organized around services/projects
  - what does this sound similar to?
- operational burden is **shared** by the developers who are building the system
  - better alignment of incentives between developers and operators, since same people perform both roles
- encourage operators to automate **toil**
- may still have some dedicated ops roles (e.g., SREs at Google)

Operations: the DevOps approach

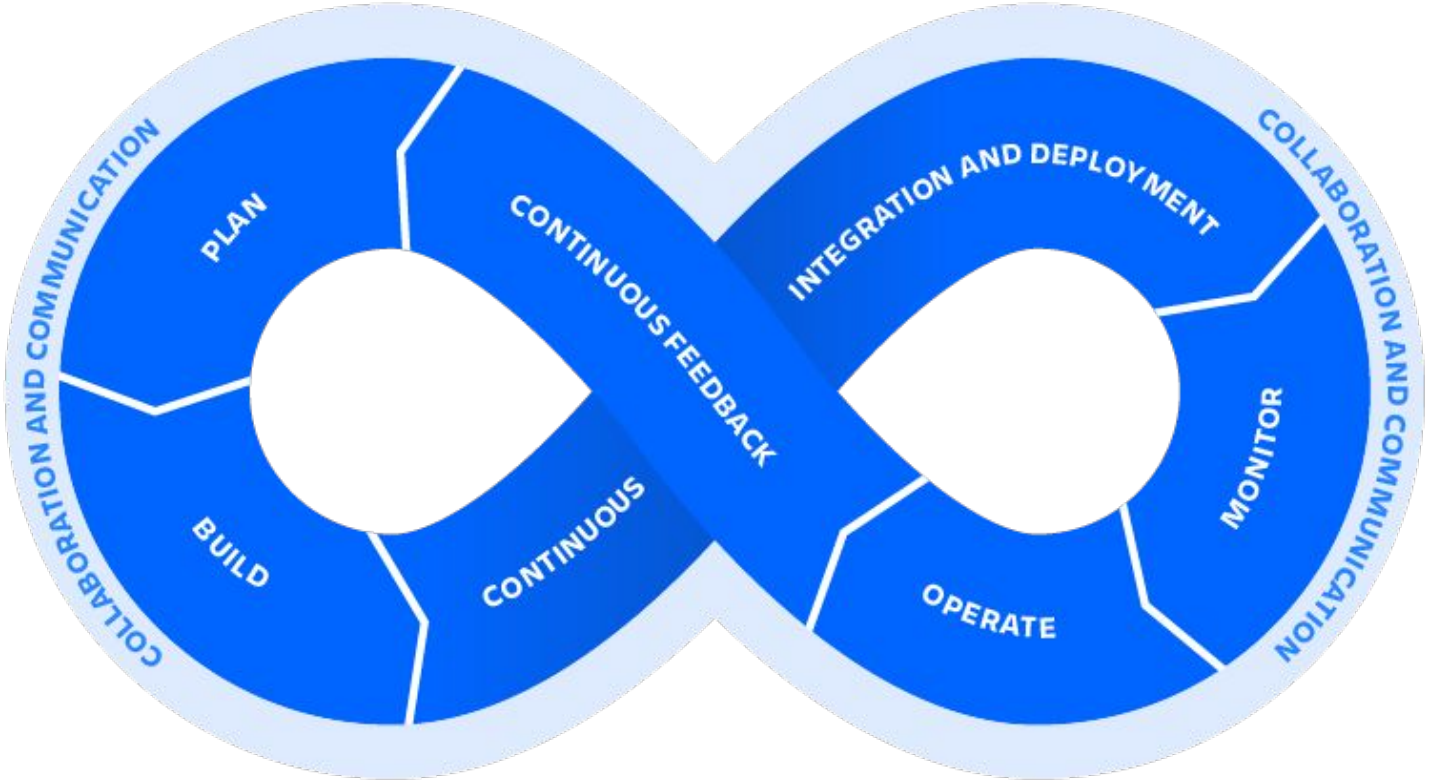

figure credit: Atlassian

Operations: toil

> *If a human operator needs to touch your system during normal operations, you have a bug. The definition of normal changes as your systems grow.*
>
> Carla Geisser, Google SRE

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

A key advantage of DevOps is that it encourages **removing** toil

- if operators are separate from devs, devs have no incentive to avoid toil

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

- **manual**: includes work such as manually running a script that automates some task (typing the command itself is toil!)

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

- **manual**: includes work such as manually running a script that automates some task (typing the command itself is toil!)
- **repetitive**: if you're performing a task for the first time ever, or even the second time, this work is not toil

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

- **manual**: includes work such as manually running a script that automates some task (typing the command itself is toil!)
- **repetitive**: if you're performing a task for the first time ever, or even the second time, this work is not toil
- **automatable**: if human judgment is essential for the task, there's a good chance it's not toil

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

- **tactical**: toil is usually interrupt-driven and reactive

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

- **tactical**: toil is usually interrupt-driven and reactive
- **no enduring value**: if your service remains in the same state after you have finished a task, the task was probably toil

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

- **tactical**: toil is usually interrupt-driven and reactive
- **no enduring value**: if your service remains in the same state after you have finished a task, the task was probably toil
- **O(n) with service growth**: if the work involved in a task scales up linearly with *service size*, *traffic volume*, or *user count*, that task is probably toil

Operations: toil

Definition: ***toil*** is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows

- **tactical**: toi
- **no enduri** e after you have fi
- **O(n) with s** les up linearly wit sk is probably toil

A task doesn't need to have **all** of these attributes to be toil. But, the more closely work matches one or more of these descriptors, the **more likely** it is to be toil.

Operations: toil

Things that **aren’t** toil:

Operations: toil

Things that **aren’t** toil:

- work you don’t like to do is **not always** toil

Operations: toil

Things that **aren't** toil:

- work you don't like to do is **not always** toil
  - useful, productive work can be unpleasant
    - e.g., cleaning up the entire alerting configuration for your service and removing clutter may not be fun, but it's not toil

Operations: toil

Things that **aren't** toil:

- work you don't like to do is **not always** toil
  - useful, productive work can be unpleasant
    - e.g., cleaning up the entire alerting configuration for your service and removing clutter may not be fun, but it's not toil
  - but most toil is unpleasant

Operations: toil

Things that **aren't** toil:

- work you don't like to do is **not always** toil
  - useful, productive work can be unpleasant
    - e.g., cleaning up the entire alerting configuration for your service and removing clutter may not be fun, but it's not toil
  - but most toil is unpleasant
- **overhead** is also different than toil

Operations: toil

Things that **aren't** toil:

- work you don't like to do is **not always** toil
  - useful, productive work can be unpleasant
    - e.g., cleaning up the entire alerting configuration for your service and removing clutter may not be fun, but it's not toil
  - but most toil is unpleasant
- **overhead** is also different than toil
  - tasks like team meetings, setting and grading goals, and HR paperwork (that are not tied to operations) are overhead

Operations: toil

What’s **so bad** about toil?

Operations: toil

What's **so bad** about toil?

- career stagnation (it doesn't get you promoted)
- lowers morale (it's boring)
- creates confusion (easy to forget to do a manual task!)
- slows progress (could be doing useful work instead)
- sets precedent (avoid letting toil become normal!)
- promotes attrition ("I want to work on something interesting!")

Operations: toil

What's **so bad** about toil?

- career stagnation (it doesn't get you promoted)
- lowers morale (it's boring)
- creates co
- slows prog
- sets prece
- promotes esting!")

Despite all this, a **little bit** of toil is often okay. After all, engineers only have so many productive hours in every day, and sometimes a **mental break** is nice :)

DevOps example: Google SREs

DevOps example: Google SREs

- SRE teams are a mix of:
  - software engineers
  - software-inclined sysadmins

DevOps example: Google SREs

- SRE teams are a mix of:
  - software engineers
  - software-inclined sysadmins
- goal: SRE teams should spend at least 50% of their time on “development” work and at most 50% on toil

DevOps example: Google SREs

- SRE teams are a mix of:
  - software engineers
  - software-inclined sysadmins
- goal: SRE teams should spend at least 50% of their time on “development” work and at most 50% on toil
- SRE teams are assigned to a collection of related “SWE” (i.e., software engineering/development) teams, each of which works on one of the systems
  - SRE team manages ops for all of these systems

DevOps example: Google SREs

- SRE teams are a mix of:
  - software engineers
  - software-inclined sysadmins
- goal: SRE teams should spend at least 50% of their time on “development” work and at most 50% on toil
- SRE teams are assigned to a collection of related “SWE” (i.e., software engineering/development) teams, each of which works on one of the systems
  - SRE team manages ops for all of these systems
- SRE motto: “Hope is not a strategy”

Another DevOps example: AWS

Another DevOps example: AWS

- unlike Google, AWS does **not** have dedicated ops teams

Another DevOps example: AWS

- unlike Google, AWS does **not** have dedicated ops teams
- all development teams are **solely responsible** for the operations of their own services

Another DevOps example: AWS

- unlike Google, AWS does **not** have dedicated ops teams
- all development teams are **solely responsible** for the operations of their own services
  - teams are also small (“two-pizza”) and usually organized around a single microservice

Another DevOps example: AWS

- unlike Google, AWS does **not** have dedicated ops teams
- all development teams are **solely responsible** for the operations of their own services
  - teams are also small (“two-pizza”) and usually organized around a single microservice
- this setup is **leaner** (no need to staff SRE teams!)

Another DevOps example: AWS

- unlike Google, AWS does **not** have dedicated ops teams
- all development teams are **solely responsible** for the operations of their own services
  - teams are also small (“two-pizza”) and usually organized around a single microservice
- this setup is **leaner** (no need to staff SRE teams!)
  - but means teams must **choose** between delivering new features and reducing operational burden

Another DevOps example: AWS

- unlike Google, AWS does **not** have dedicated ops teams
- all development teams are **solely responsible** for the operations of their own services
  - teams are also small (“two-pizza”) and usually organized around a single microservice
- this setup is **leaner** (no need to staff SRE teams!)
  - but means teams must **choose** between delivering new features and reducing operational burden
    - makes technical debt riskier to take on (why?)

DevOps (1/2)

Today's agenda:

- Reading Quiz
- Operations, Toil, and the DevOps philosophy
- **Ops challenge: deployment**
- Achieving reliability
  - the service reliability hierarchy + SLAs/targets
  - monitoring
  - incident/emergency response
  - post-mortems + learning from failure

Deployment

Deployment

- **Deployment** is the process of installing your software on a server (with its dependencies), connecting it to the internet, etc.

Deployment

- **Deployment** is the process of installing your software on a server (with its dependencies), connecting it to the internet, etc.
- The key challenge in deployment is **predictability**: we want to make sure that the software behaves as expected when deployed
  - i.e., we want to avoid “it works on my machine” syndrome

Deployment

- **Deployment** is the process of installing your software on a server (with its dependencies), connecting it to the internet, etc.
- The key challenge in deployment is **predictability**: we want to make sure that the software behaves as expected when deployed
  - i.e., we want to avoid “it works on my machine” syndrome
- Other challenges:
  - may need to run on a wide variety of servers
  - may need to run on servers you don’t control/own
  - may need to safely share secrets (e.g., ssh keys)

Deployment: cloud vs on-prem

- When deploying a service, you usually have two choices:

Deployment: cloud vs on-prem

- When deploying a service, you usually have two choices:
  - host it on servers that you own and manage ("***on-prem*** deployment", short for "on premise")

Deployment: cloud vs on-prem

- When deploying a service, you usually have two choices:
  - host it on servers that you own and manage (“***on-prem*** deployment”, short for “on premise”)
  - pay someone else to host it (“***cloud*** deployment”)

Deployment: cloud vs on-prem

- When deploying a service, you usually have two choices:
  - host it on servers that you own and manage ("***on-prem*** deployment", short for "on premise")
  - pay someone else to host it ("***cloud*** deployment")
    - within the general cloud deployment category, you may get to choose whether to rent whole servers, share time on servers, or even pretend not to have a server at all (this is called "***function-as-a-service***", e.g., via AWS Lambda)

Deployment: cloud vs on-prem

- Advantages of on-prem deployment:

- Advantages of cloud deployment:

Deployment: cloud vs on-prem

- Advantages of on-prem deployment:
  - you have total control of the system, which might have reliability and security benefits
  - can choose exactly the right hardware
  - no “vendor lock-in”
- Advantages of cloud deployment:
  - cloud providers usually have better ops than you do
  - ability to add more servers quickly (“**auto-scaling**”)
  - easy access to datacenters in multiple regions

Deployment: installing software

Deployment: installing software

- Directly installing your software onto the machine's main operating system (***bare metal*** deployment) is **rare** (especially when deploying into the cloud)

Deployment: installing software

- Directly installing your software onto the machine's main operating system (***bare metal*** deployment) is **rare** (especially when deploying into the cloud)
  - ideally, you want all of your servers to have the **same environment** (so that if there is a problem, you only need to debug it in one context)

Deployment: installing software

- Directly installing your software onto the machine's main operating system (***bare metal*** deployment) is **rare** (especially when deploying into the cloud)
  - ideally, you want all of your servers to have the **same environment** (so that if there is a problem, you only need to debug it in one context)
  - in practice, this is achieved via ***virtualization***

Deployment: virtualization

Definition: ***virtualization*** is the use of software to simulate portions of a computer system

Deployment: virtualization

Definition: ***virtualization*** is the use of software to simulate portions of a computer system

- we can use virtualization to present a system that **appears the same** to our software, regardless of the underlying hardware

Deployment: virtualization

Definition: ***virtualization*** is the use of software to simulate portions of a computer system

- we can use virtualization to present a system that **appears the same** to our software, regardless of the underlying hardware
- three major kinds:
  - Full virtualization (a.k.a. ***emulation***)
  - ***Paravirtualization***/OS virtualization
  - ***Container*** virtualization

Deployment: virtualization: emulation

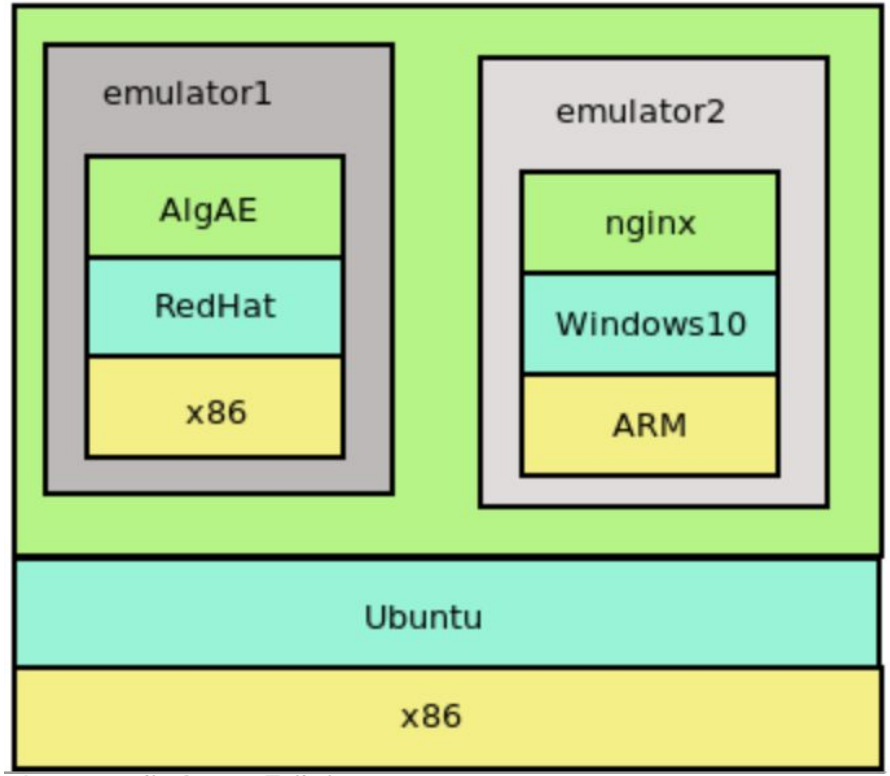

Image credit: Steven Zeil via
https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: emulation

- The host **simulates everything** down to and including the CPU level of the guest

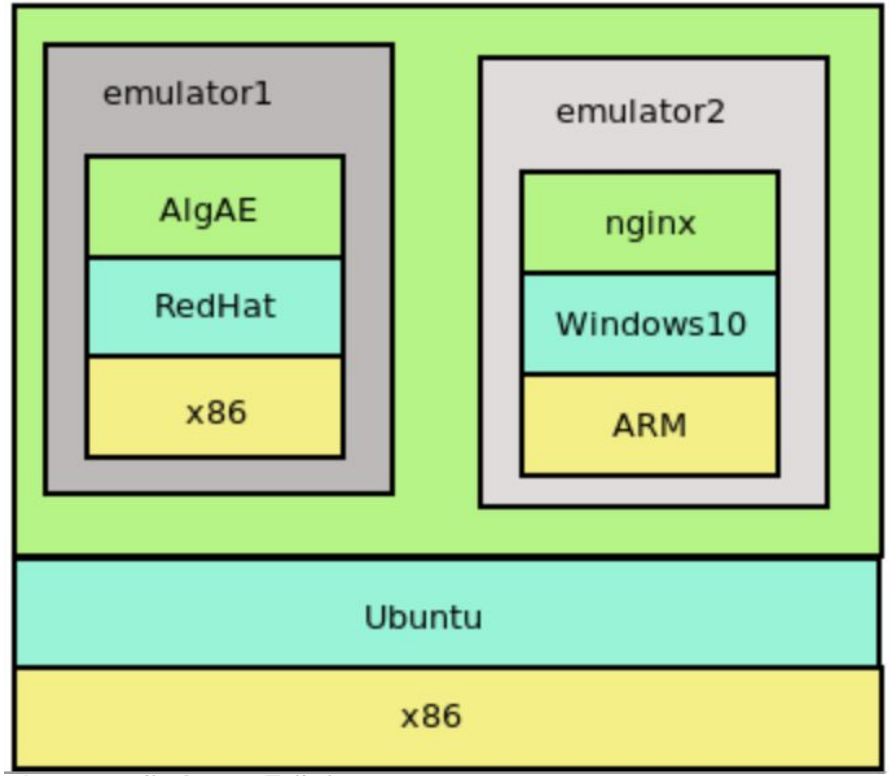

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: emulation

- The host **simulates everything** down to and including the CPU level of the guest
- The guest CPU can be different from that of the host

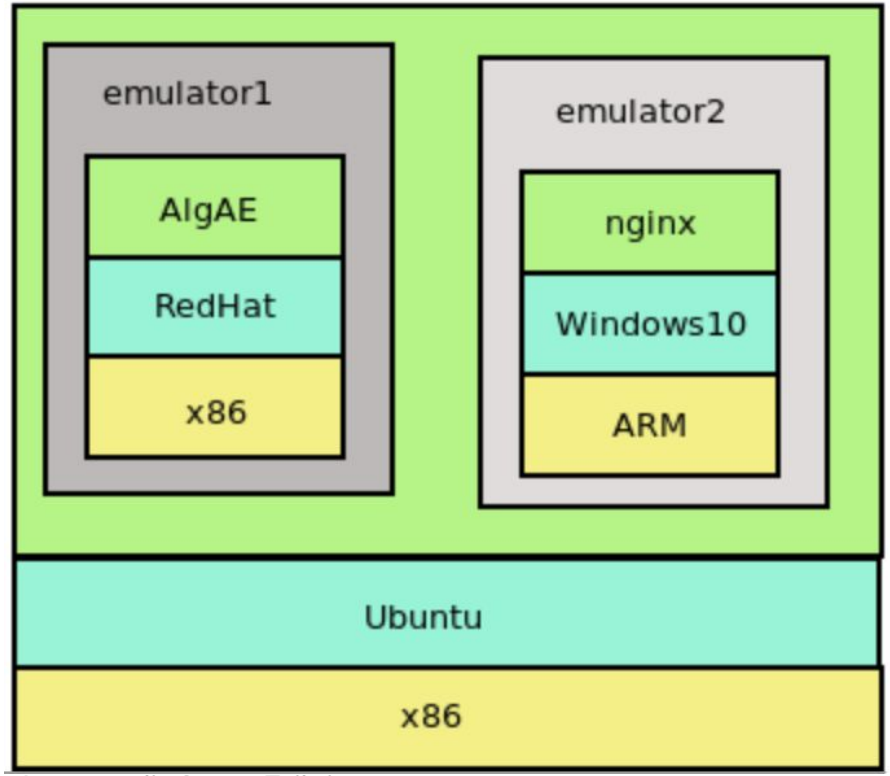

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: emulation

- The host **simulates everything** down to and including the CPU level of the guest
- The guest CPU can be different from that of the host
- Examples: the JVM, game console emulators

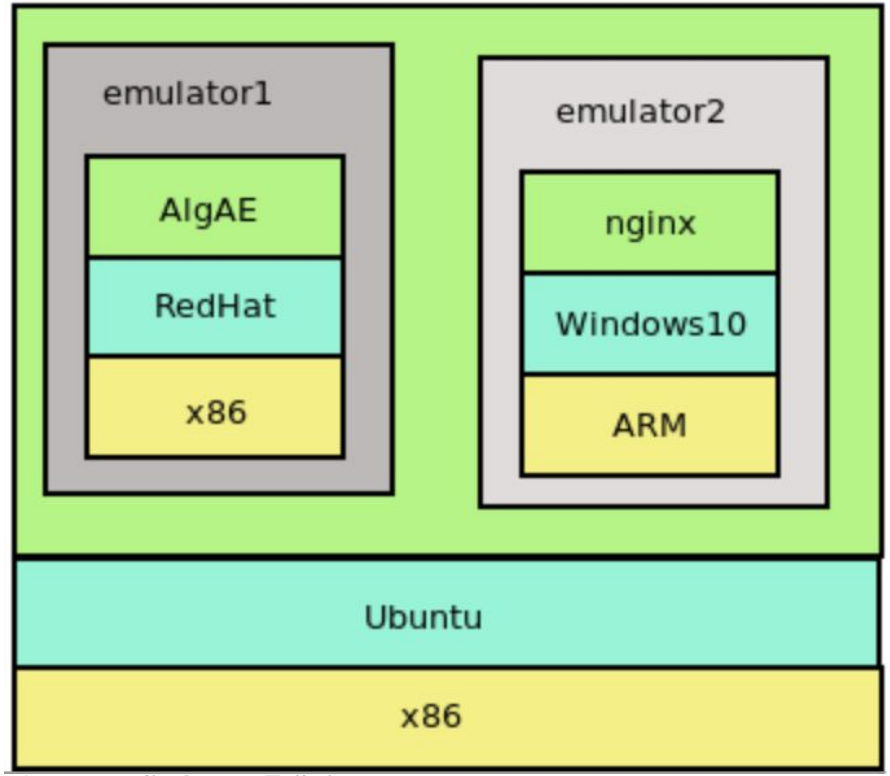

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

Deployment: virtualization: paravirtualization

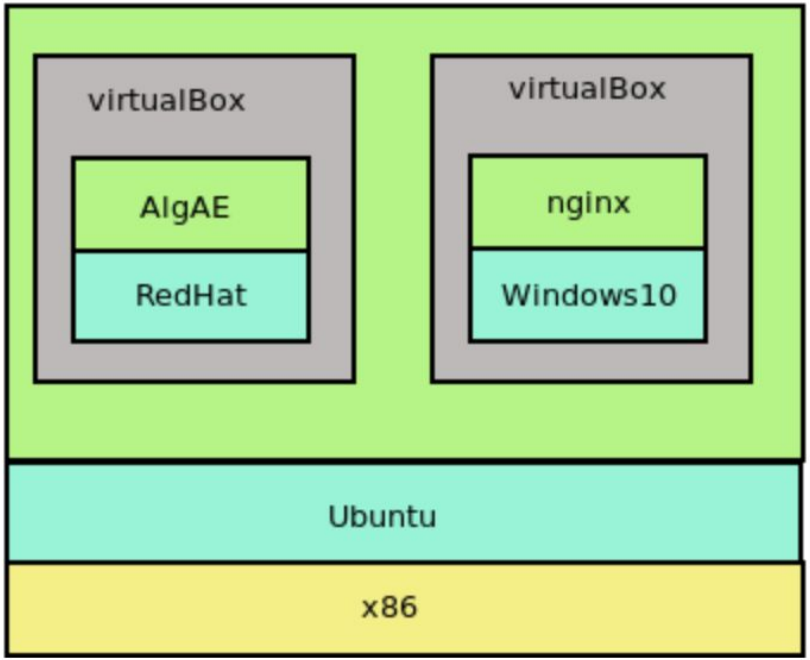

Image credit: Steven Zeil via
https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- CPU is not emulated, but OS is

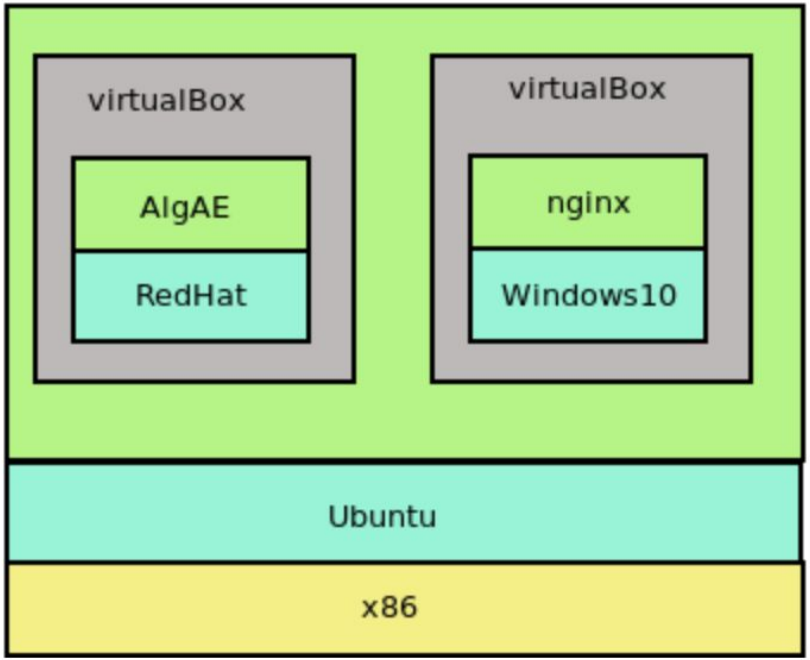

Image credit: Steven Zeil via
https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- CPU is not emulated, but OS is
- Allows code between OS calls to run natively

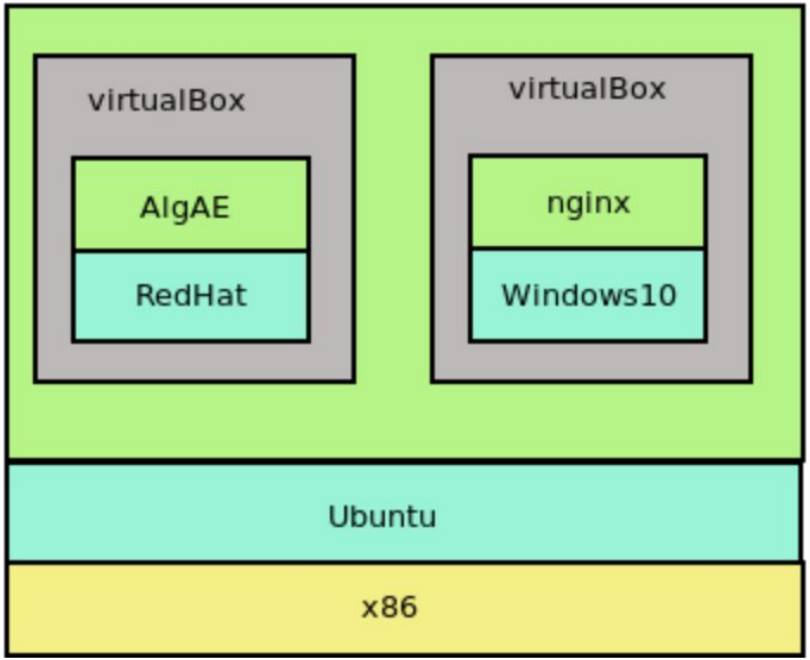

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- CPU is not emulated, but OS is
- Allows code between OS calls to run natively
- Many devices are simulated

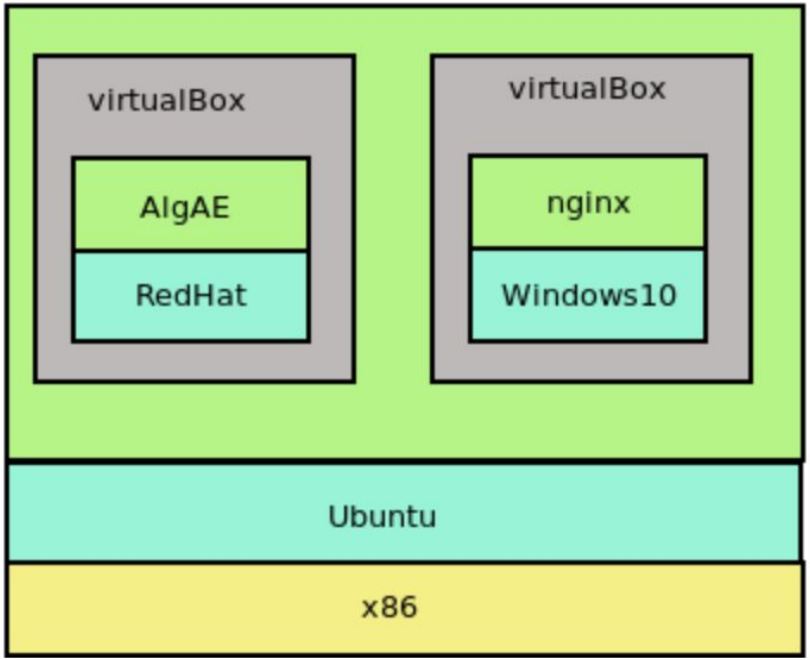

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- CPU is not emulated, but OS is
- Allows code between OS calls to run natively
- Many devices are simulated
- Guest CPU must be same as on host

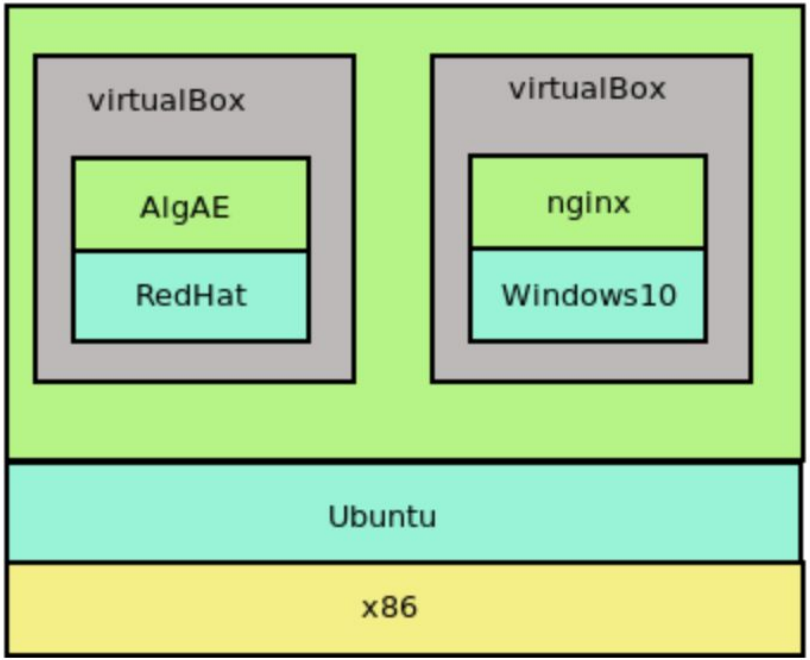

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- CPU is not emulated, but OS is
- Allows code between OS calls to run natively
- Many devices are simulated
- Guest CPU must be same as on host
- Guest OS can be different from that of host

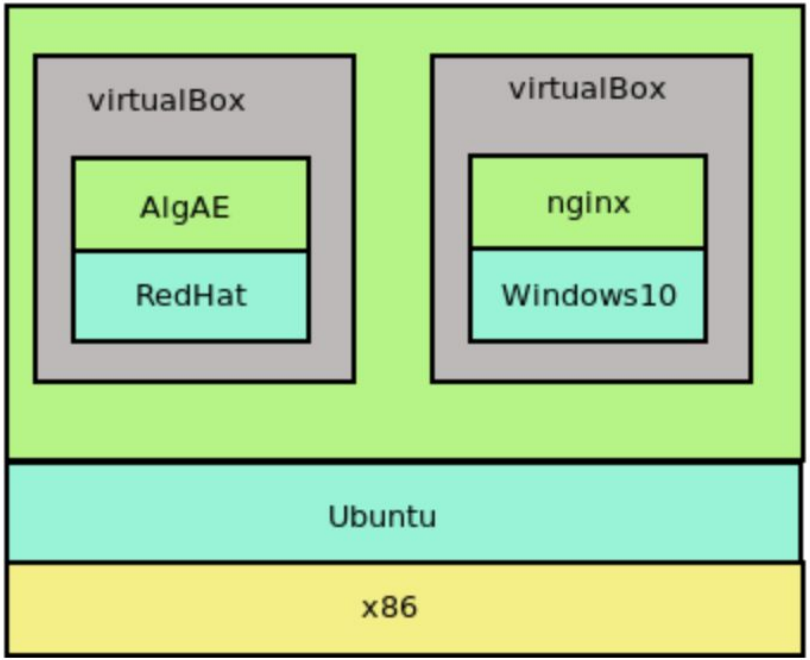

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- CPU is not emulated, but OS is
- Allows code between OS calls to run natively
- Many devices are simulated
- Guest CPU must be same as on host
- Guest OS can be different from that of host
- examples: VirtualBox, VMWare

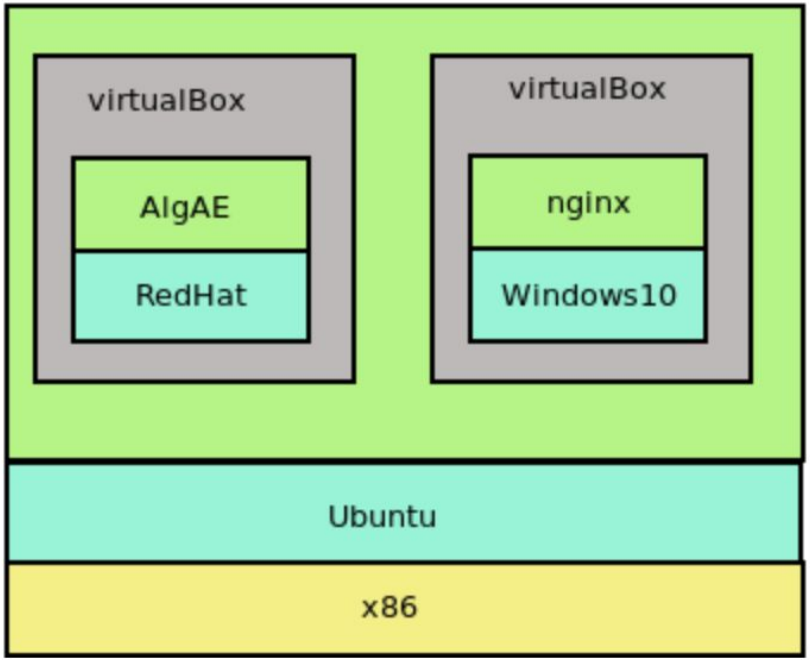

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- CPU is not emulated, but OS is
- Allows code between OS calls to run na
- Many
- Guest
  host
- Guest OS can be different from that of host
- examples: VirtualBox, VMWare

As described so far, this is ***hosted*** paravirtualization, where the “virtual machine” is just a user-space program.

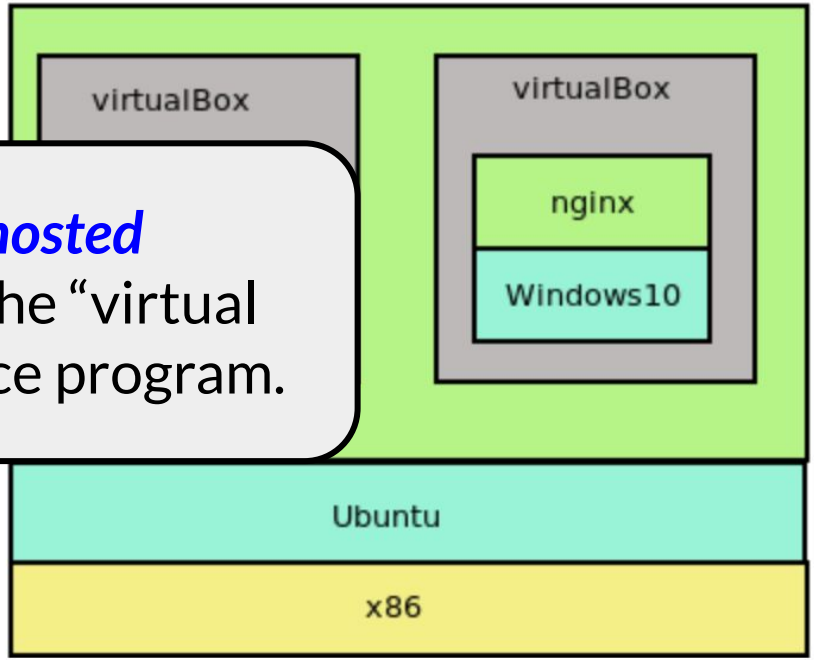

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- Alternative to hosted paravirtualization: ***hypervisors***

Deployment: virtualization: paravirtualization

- Alternative to hosted paravirtualization: ***hypervisors***
- A hypervisor is a special “thin” operating system that runs other operating systems

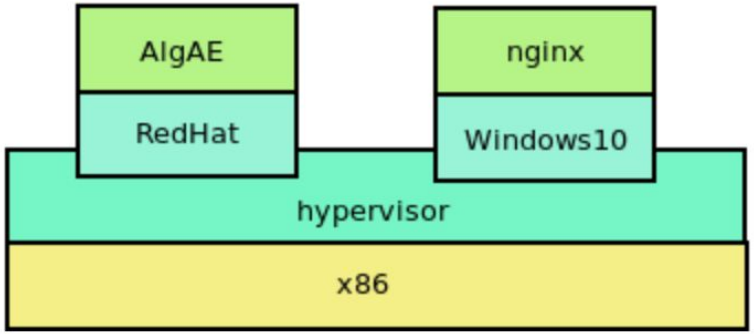

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- Alternative to hosted paravirtualization: ***hypervisors***
- A hypervisor is a special “thin” operating system that runs other operating systems
  - this is how cloud machines actually are deployed

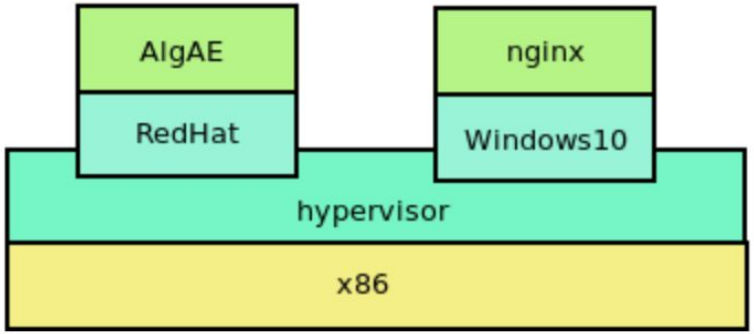

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: paravirtualization

- Alternative to hosted paravirtualization: ***hypervisors***
- A hypervisor is a special “thin” operating system that runs other operating systems
  - this is how cloud machines actually are deployed
- Examples: Xen, Hyper-V

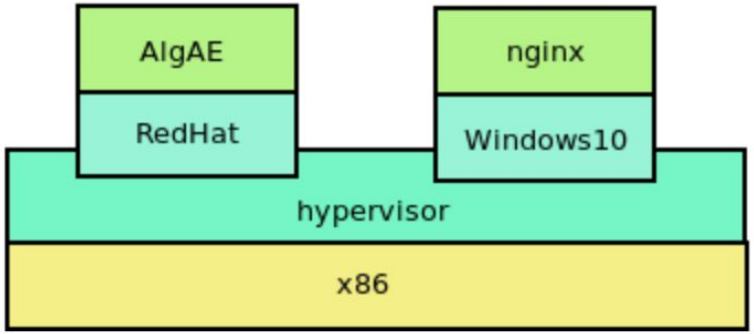

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: containers

Deployment: virtualization: containers

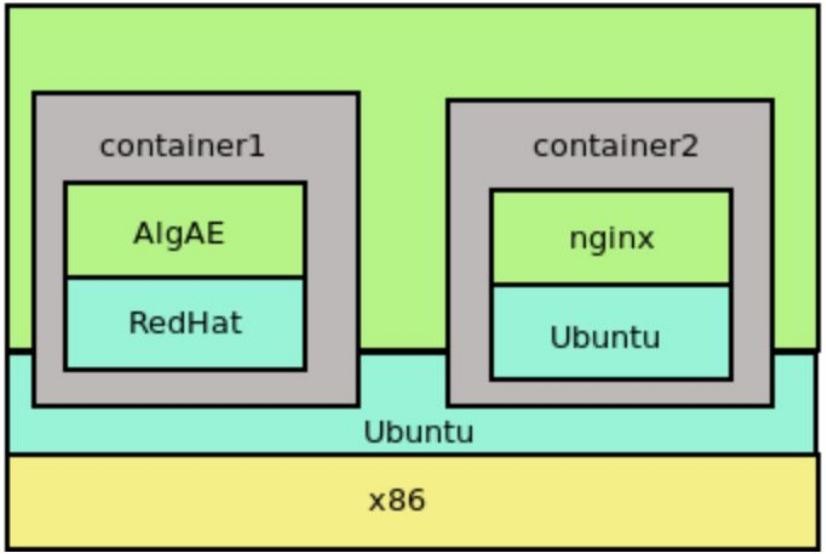

Image credit: Steven Zeil via
https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: containers

- Guest CPU is same as host's

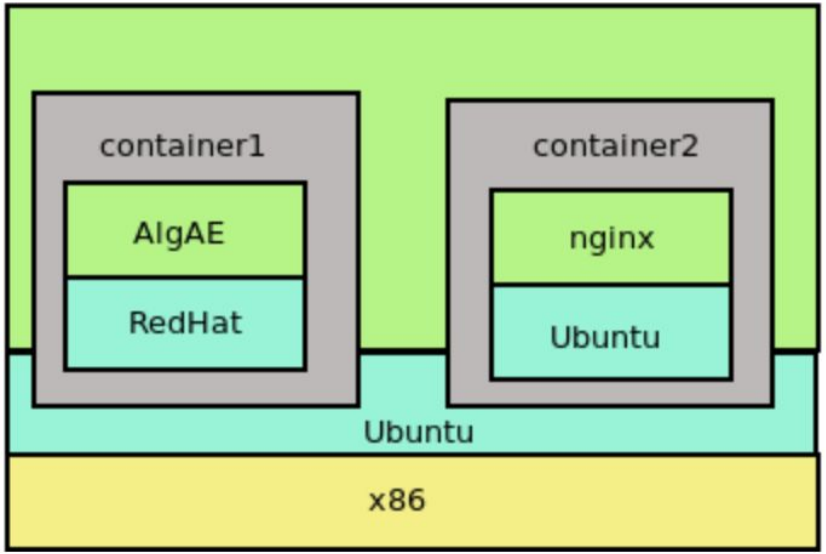

Image credit: Steven Zeil via
https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: containers

- Guest CPU is same as host's
- Guest OS is same family as host's
  - e.g., both are Linux distros

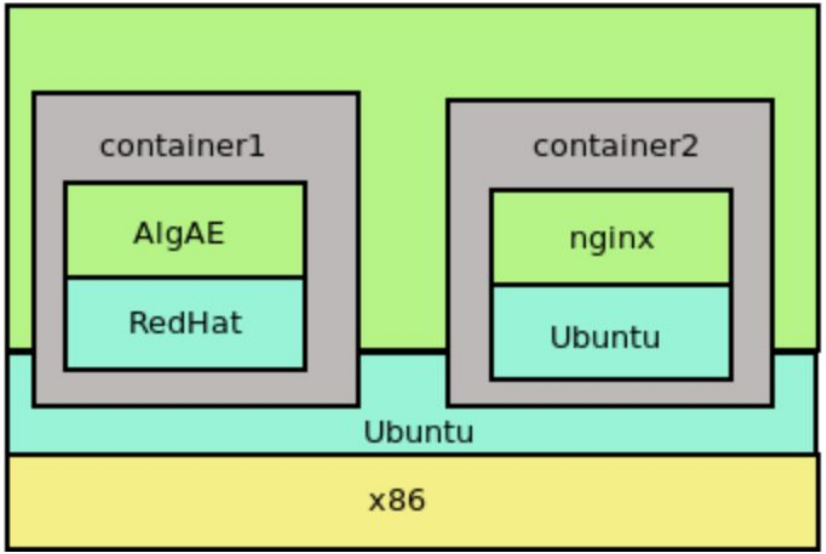

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: containers

- Guest CPU is same as host's
- Guest OS is same family as host's
  - e.g., both are Linux distros
- A thin OS simulation passes OS functions down to host.

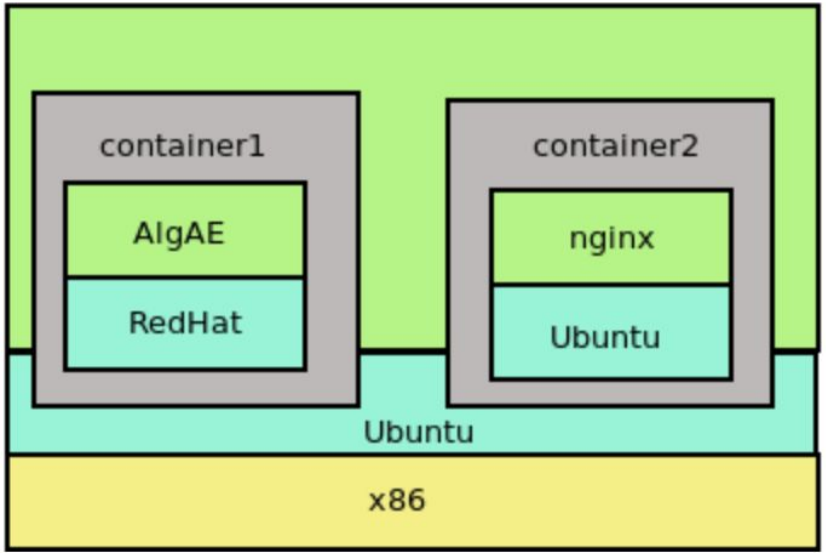

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: containers

- Guest CPU is same as host's
- Guest OS is same family as host's
  - e.g., both are Linux distros
- A thin OS simulation passes OS functions down to host.
- **Applications** are simulated

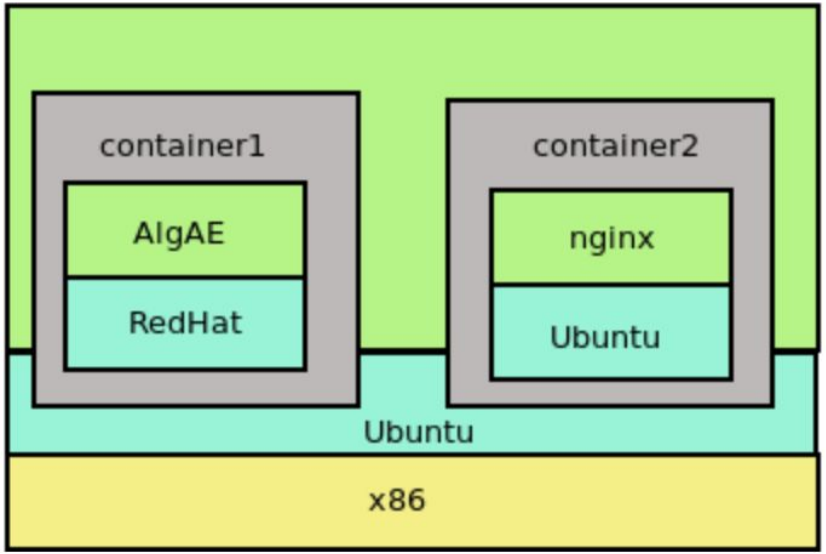

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: containers

- Guest CPU is same as host's
- Guest OS is same family as host's
  - e.g., both are Linux distros
- A thin OS simulation passes OS functions down to host.
- **Applications** are simulated
- Examples: Docker, Kubernetes

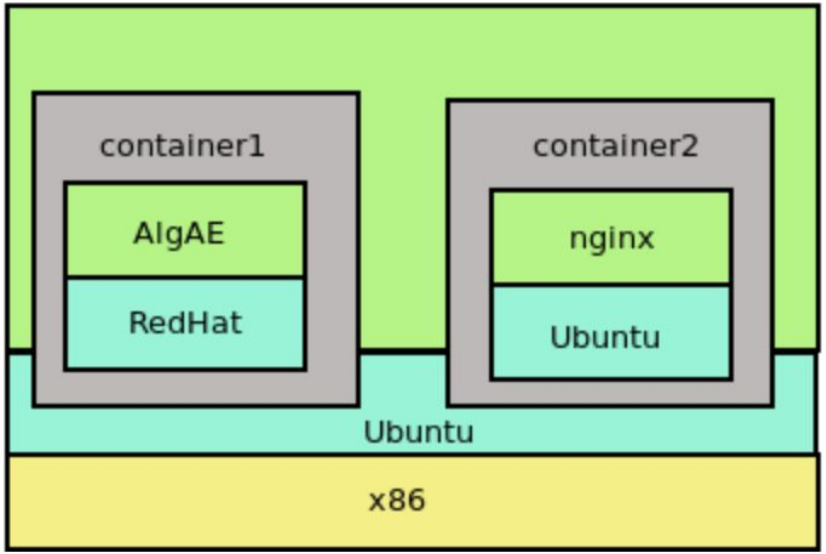

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: containers

- Guest CPU is same as host's
- Guest OS is same family as host's
  - e.g.
- A thin layer translates system calls/functions down to host.
- **Applications** are simulated
- Examples: Docker, Kubernetes

Pop Quiz: can Docker running on Windows virtualize an application written for Linux?

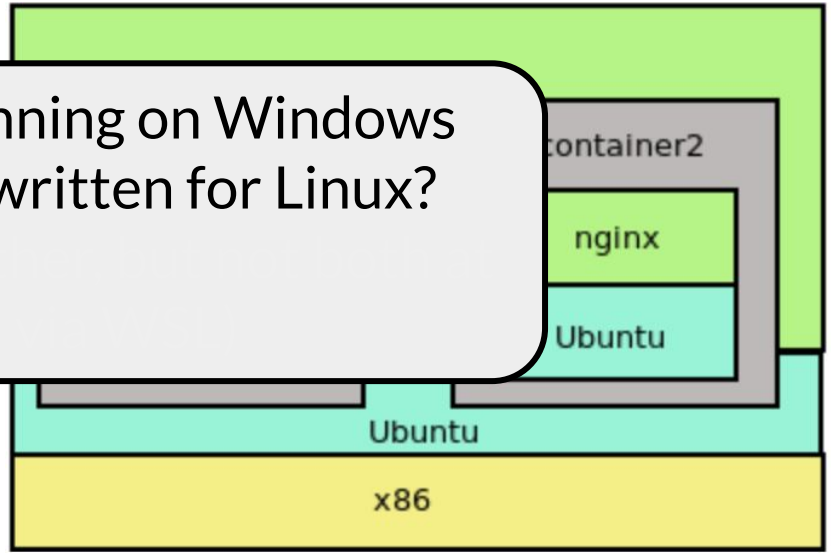

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: virtualization: containers

- Guest CPU is same as host's
- Guest OS is same family as host's
  - e.g.
- A thin layer translates system functions down to host.
- **Applications** are simulated
- Examples: Docker, Kubernetes

Pop Quiz: can Docker running on Windows virtualize an application written for Linux?

- Answer: one or the other, but not both at the same time (works via WSL)

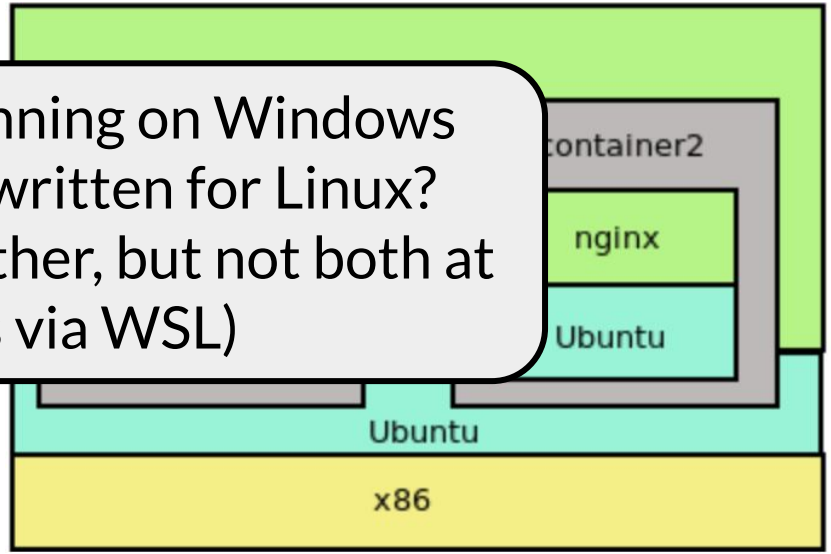

Image credit: Steven Zeil via https://www.cs.odu.edu/~zeil/cs-devops/f20/Public/virtualization/index.html

Deployment: implications

Deployment: implications

- you'll have **different options** for debugging + incident response depending on your team's virtualization strategy

Deployment: implications

- you'll have **different options** for debugging + incident response depending on your team's virtualization strategy
  - if you're using virtualization, you ought to have easy access to an identical environment for debugging. Use it!

Deployment: implications

- you'll have **different options** for debugging + incident response depending on your team's virtualization strategy
  - if you're using virtualization, you ought to have easy access to an identical environment for debugging. Use it!
- containers are **extremely useful in practice**, but limit your choice of hardware + OS

Deployment: implications

- you'll have **different options** for debugging + incident response depending on your team's virtualization strategy
  - if you're using virtualization, you ought to have easy access to an identical environment for debugging. Use it!
- containers are **extremely useful in practice**, but limit your choice of hardware + OS
  - almost all servers are some variant of Linux, so this limitation is less important in practice than it might seem

Deployment: implications

- you'll have **different options** for debugging + incident response depending on your team's virtualization strategy
  - if you're using virtualization, you ought to have easy access to an identical environment for debugging. Use it!
- containers are **extremely useful in practice**, but limit your choice of hardware + OS
  - almost all servers are some variant of Linux, so this limitation is less important in practice than it might seem
  - most big tech companies use containers in some form