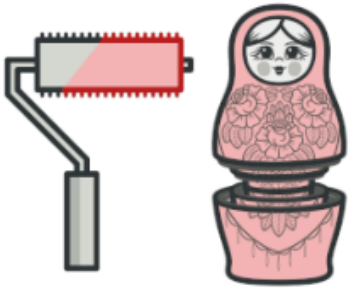
CSci 3081W: Program Design and Development

Lecture 12 – Decorator Pattern
Docker

Structural Design Patterns

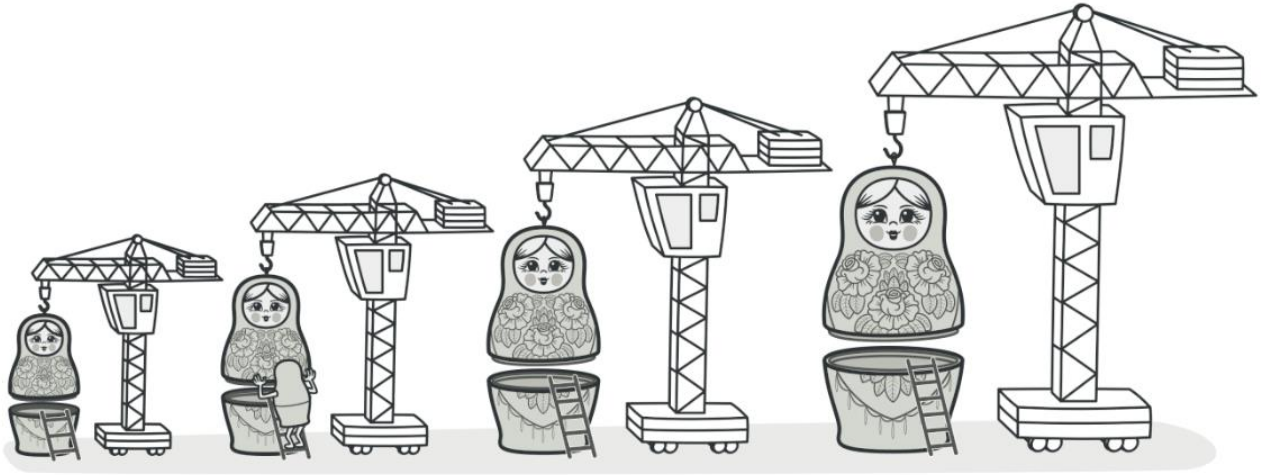


Decorator

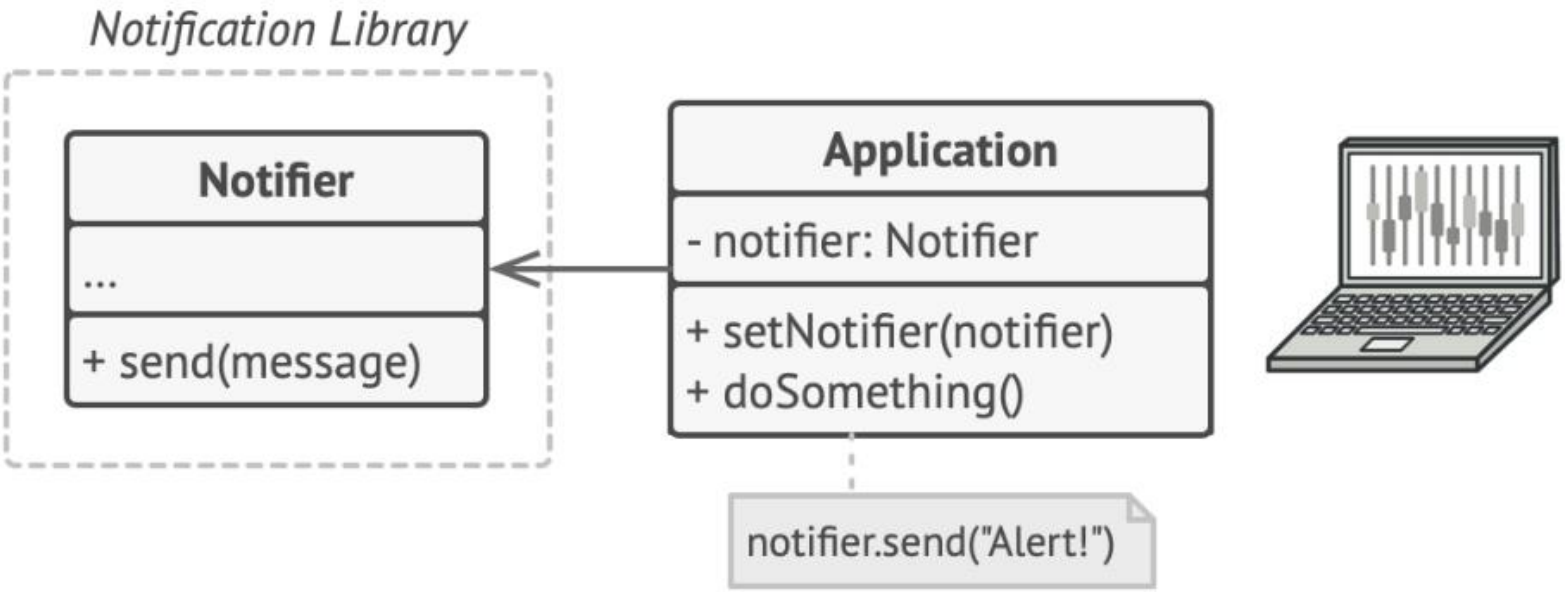
Lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.

Decorator Pattern

Decorator is a structural design pattern that lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.

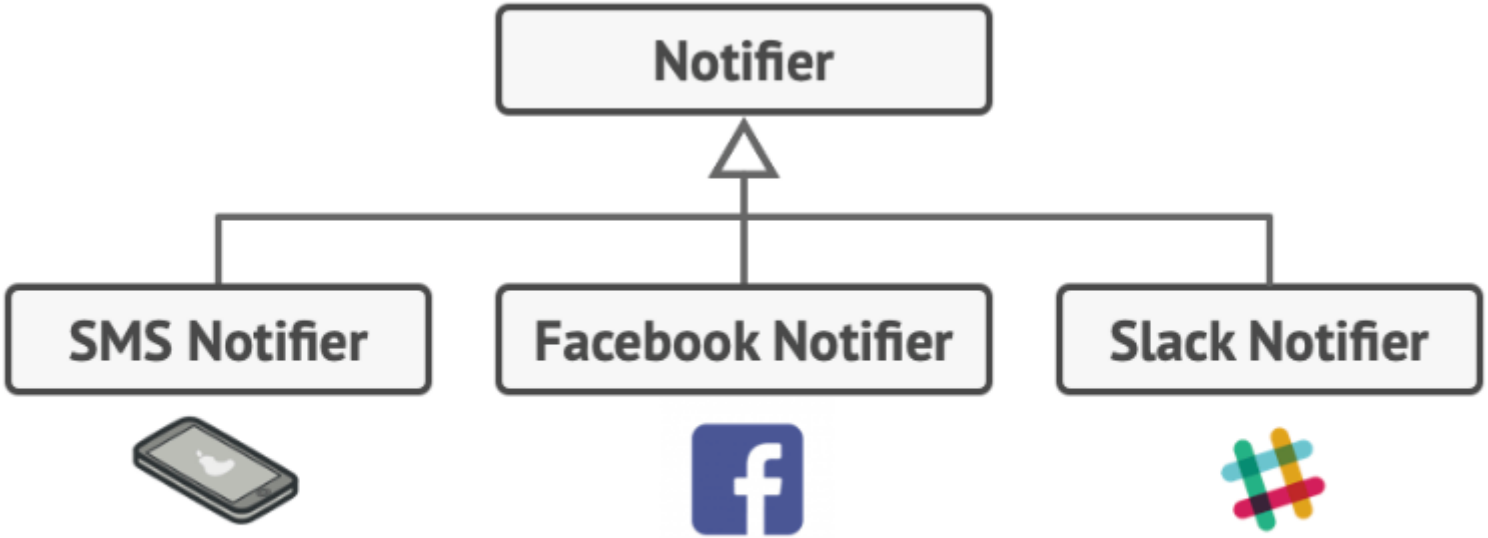


Problem



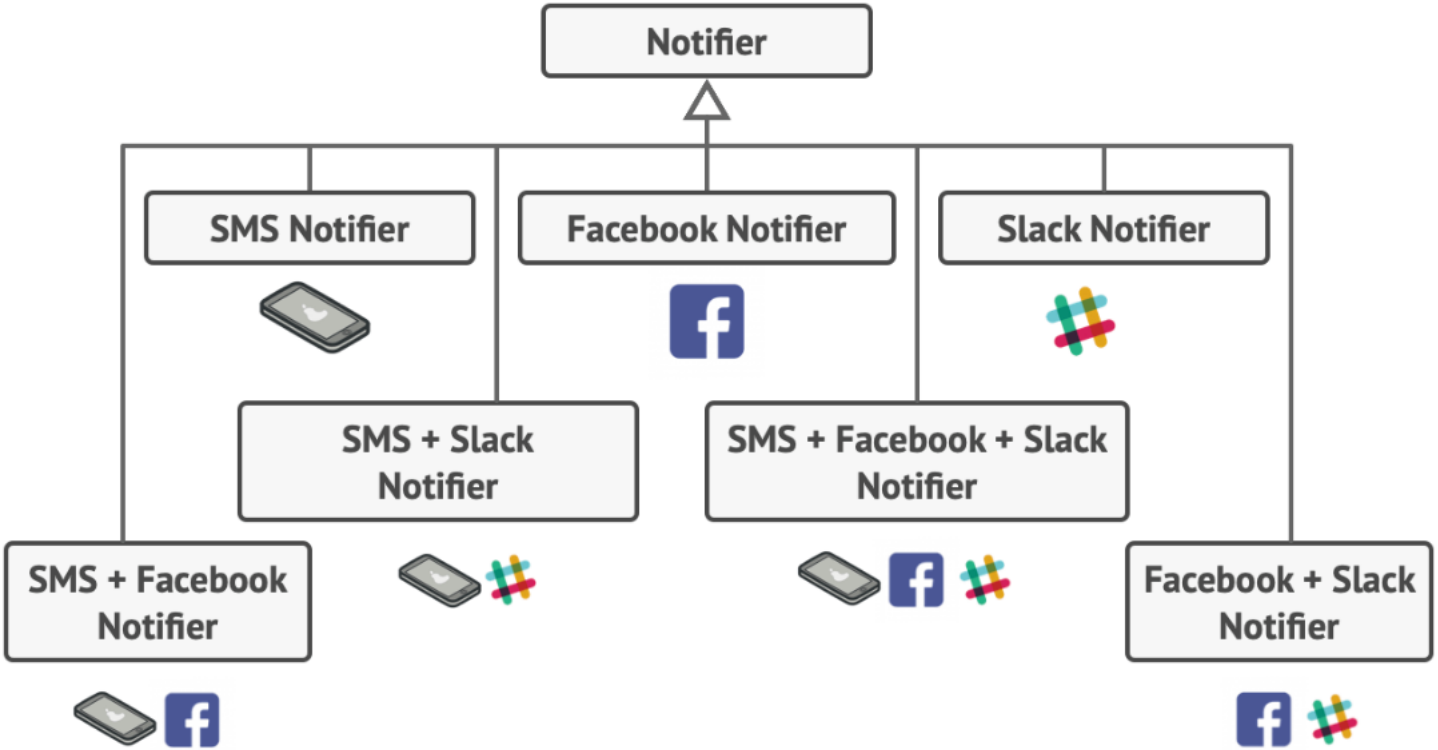
A program could use the notifier class to send notifications about important events to a predefined set of emails.

Problem



Each notification type is implemented as a notifier's subclass.

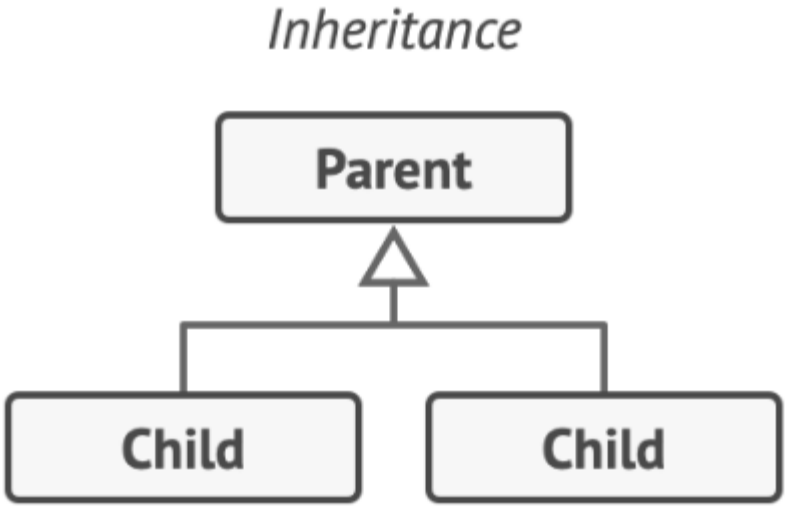
Problem



Combinatorial explosion of subclasses.

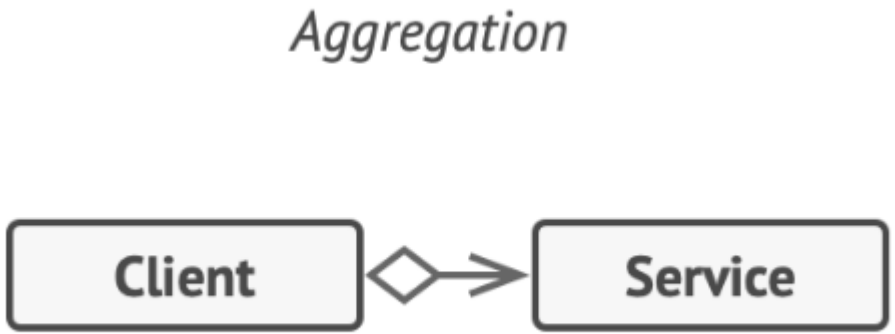
Potential Solution

Use inheritance (naïve)



Better Solution

Use aggregation



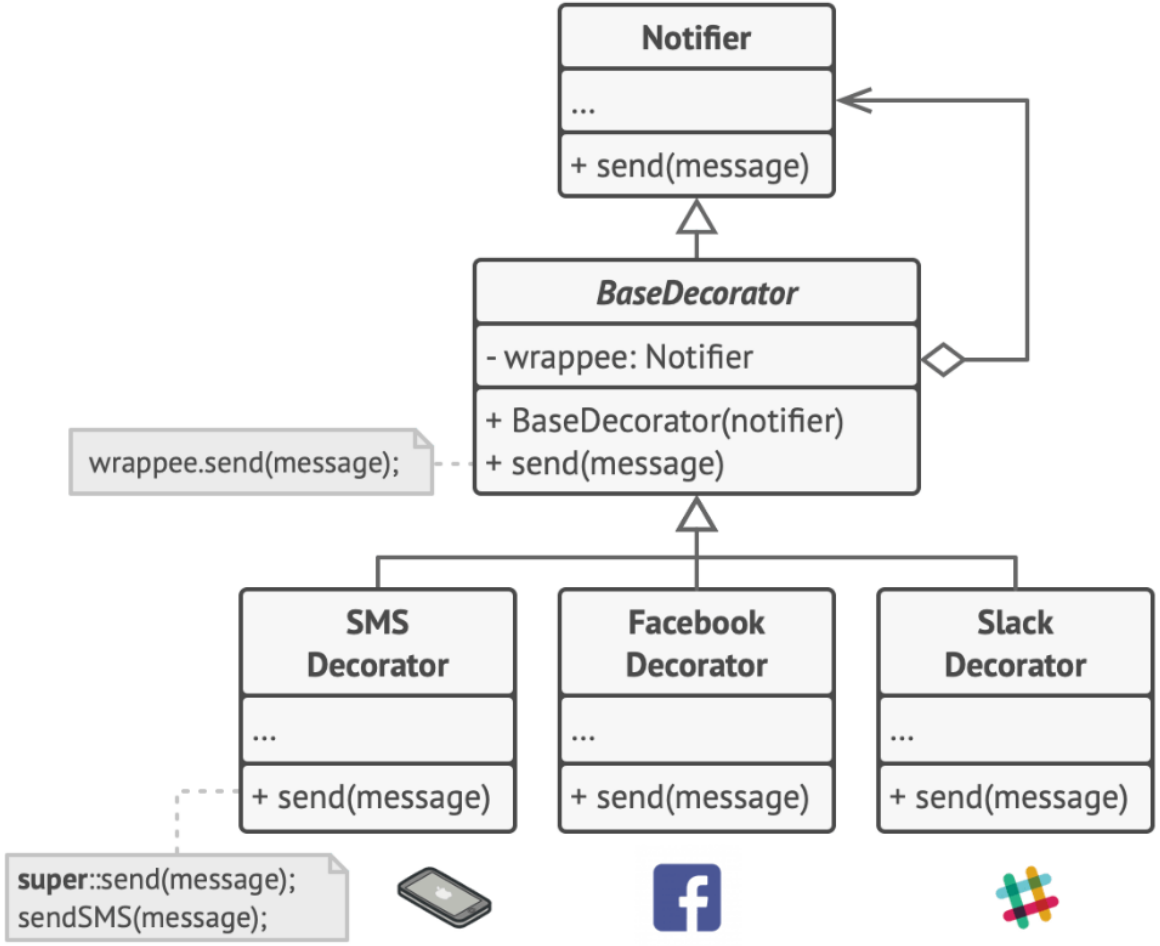
Why aggregation over inheritance?

Inheritance is static.

- You won’t be able to change behavior at runtime, only replace the whole object with a different subclass.

Also, in some languages, you can only inherit from one parent.

Solution

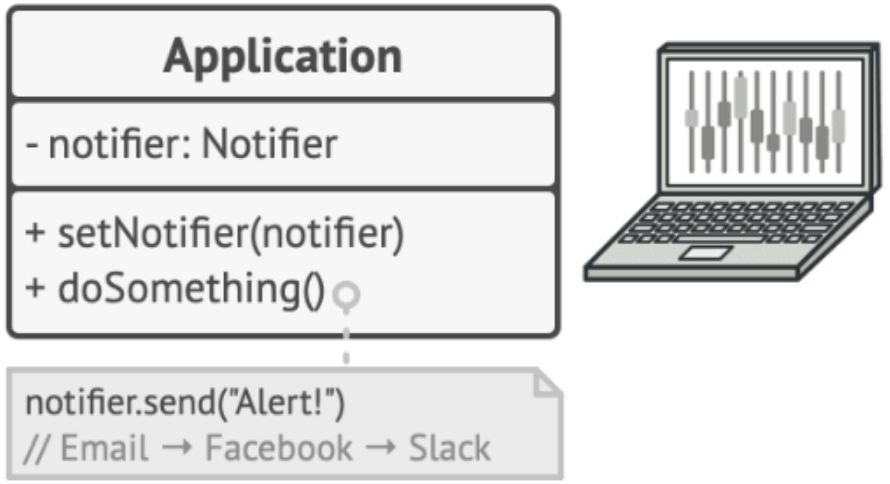


Various notification methods become decorators.

Solution

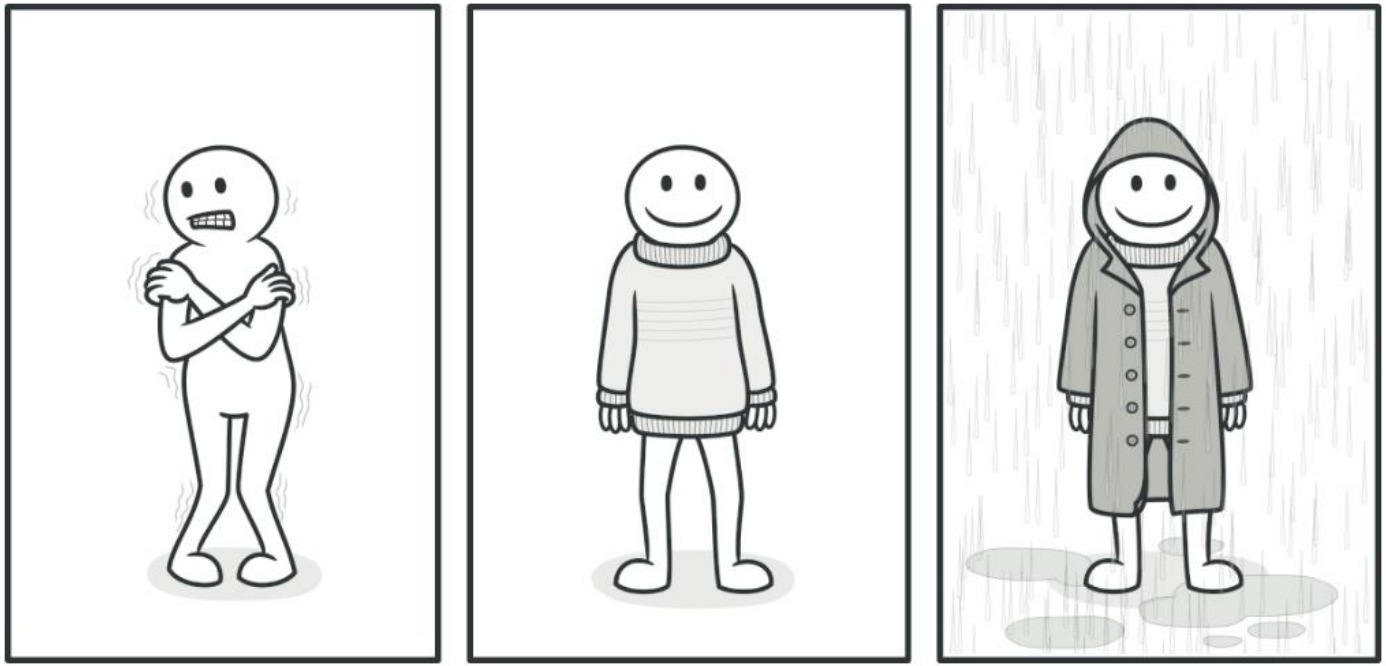
```
stack = new Notifier()
if (facebookEnabled)
    stack = new FacebookDecorator(stack)
if (slackEnabled)
    stack = new SlackDecorator(stack)

app.setNotifier(stack)
```



Apps might configure complex stacks of notification decorators.

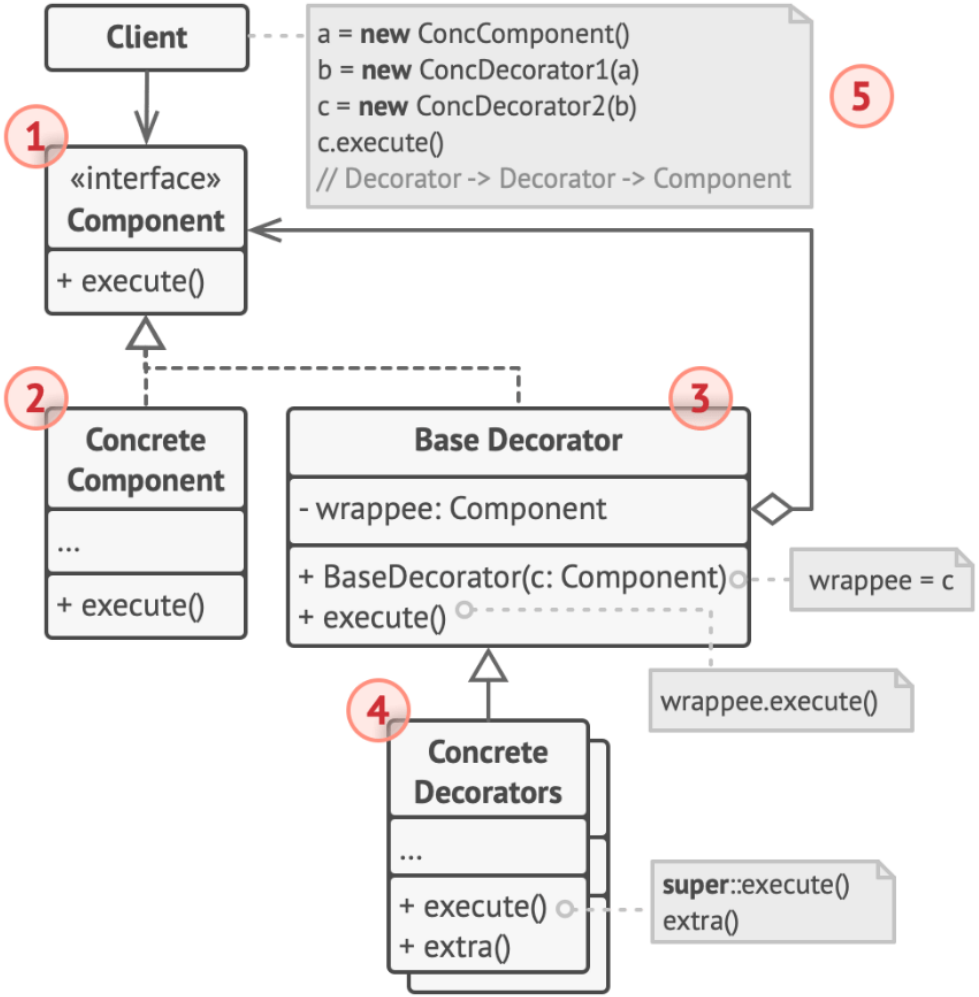
Solution



You get a combined effect from wearing multiple pieces of clothing.

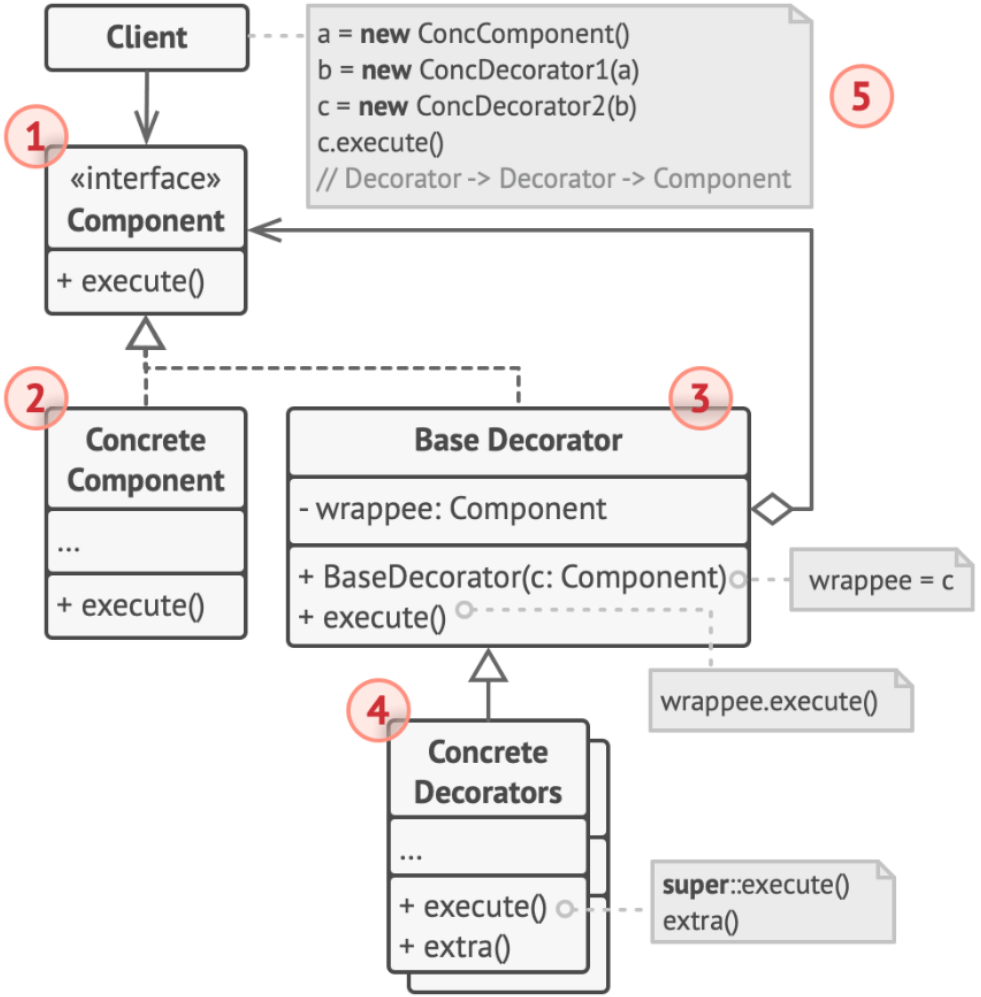
UML Class Diagram

1. The **Component** declares the common interface for both wrappers and wrapped objects.



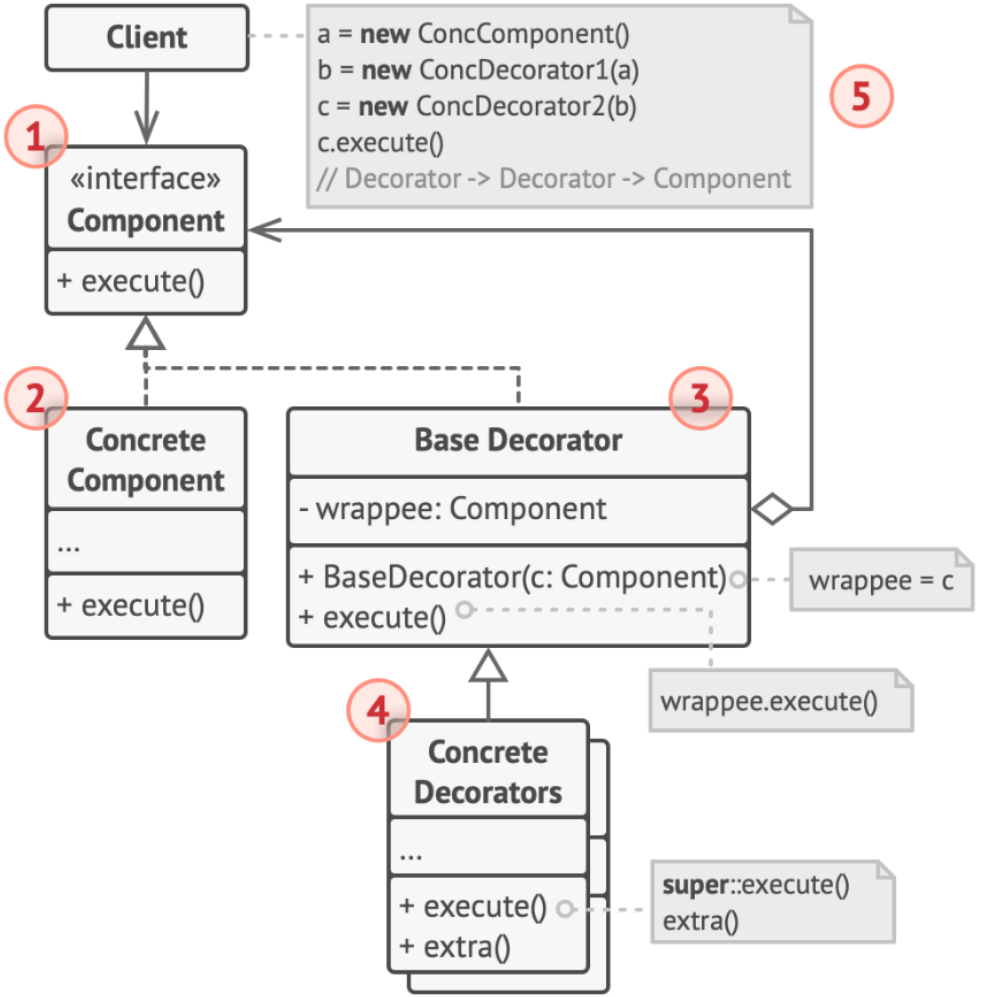
UML Class Diagram

2. **Concrete Component** is a class of objects being wrapped. It defines the basic behavior, which can be altered by decorators.



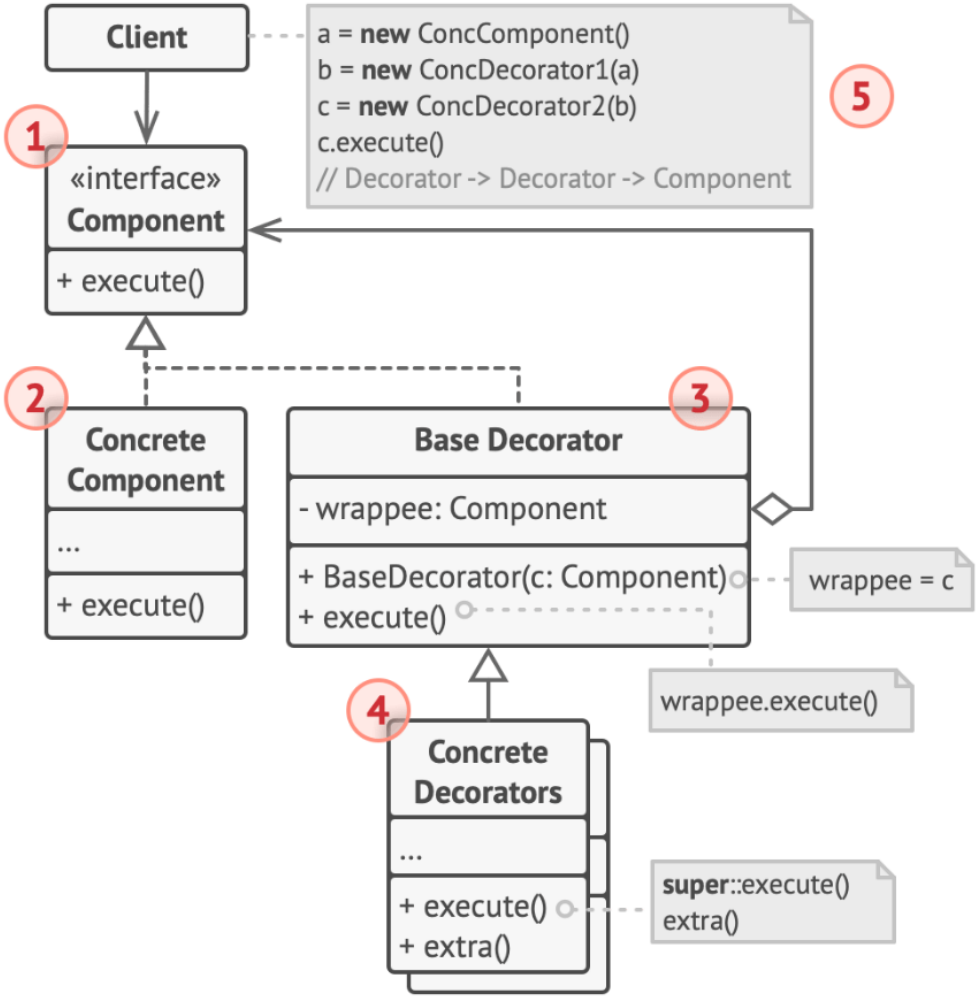
UML Class Diagram

3. The **Base Decorator** class has a field for referencing a wrapped object. The field's type is the component interface so it can contain both concrete components and decorators. The base decorator delegates all operations to the wrapped object.



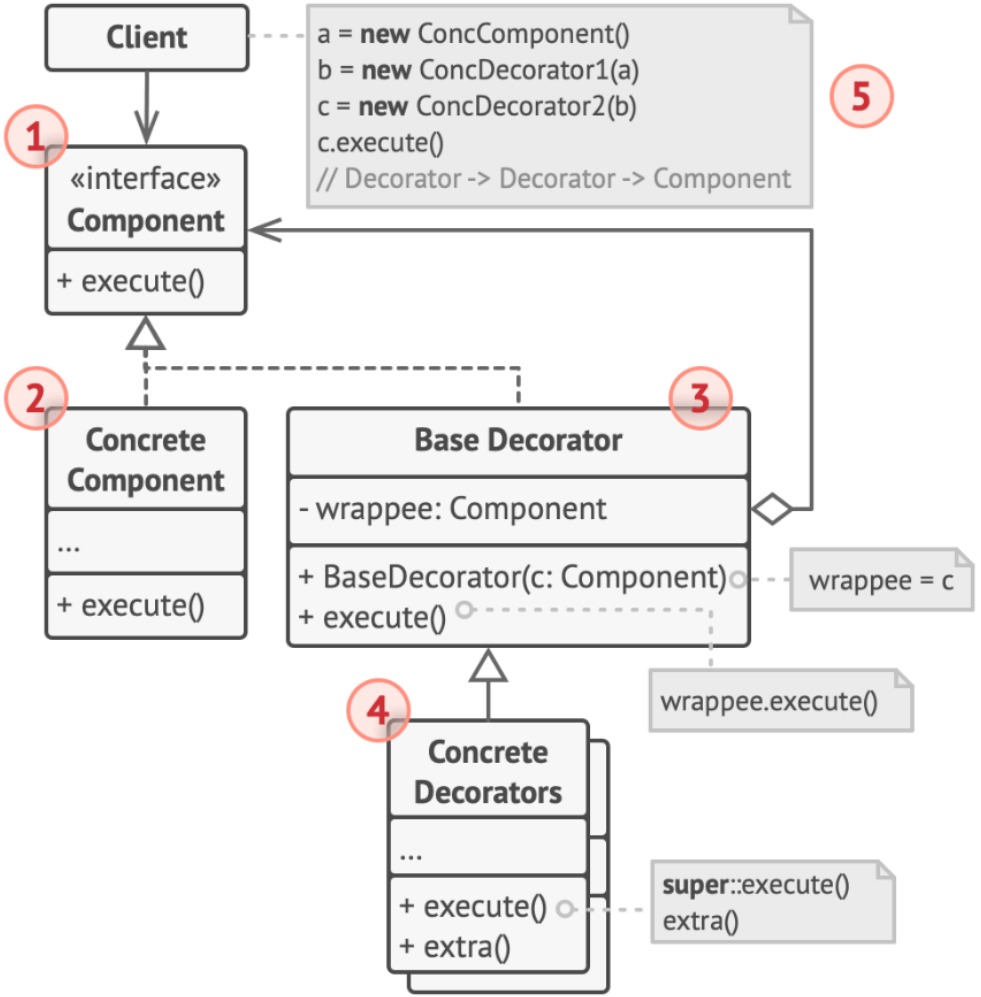
UML Class Diagram

4. **Concrete Decorators** define extra behaviors that can be added to components dynamically. Concrete decorators override methods of the base decorator and execute their behavior either before or after calling the parent method.



UML Class Diagram

5. The **Client** can wrap components in multiple layers of decorators, if it works with all objects via the component interface.



Applicability

When you want to give extra behaviors to objects at runtime without breaking any existing code

When it’s not possible to extend an object’s behavior via inheritance

The big pros

S from SOLID

Extend behavior without making a subclass

Can add and remove functionality from an object at runtime

Can combine several behaviors all into one object (using multiple decorators)

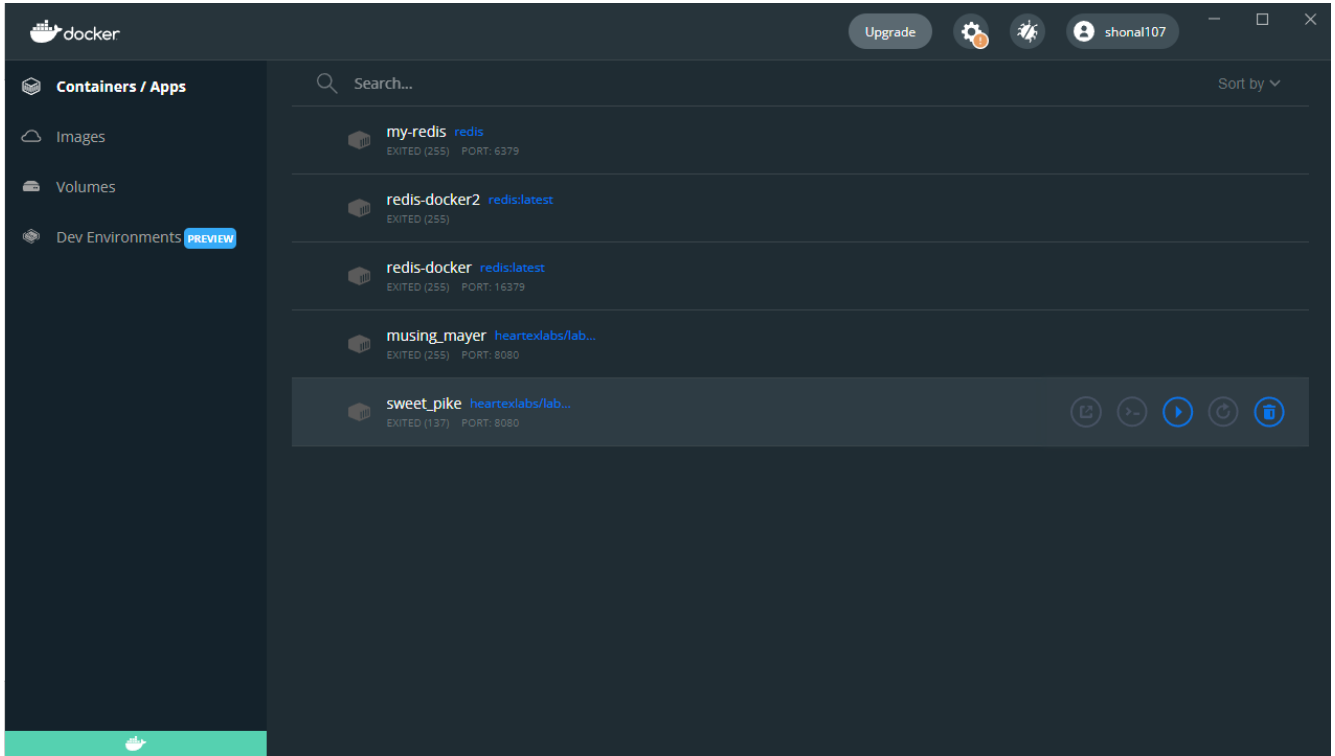
Relationships to other patterns

Composite and Decorator both use this idea of recursion

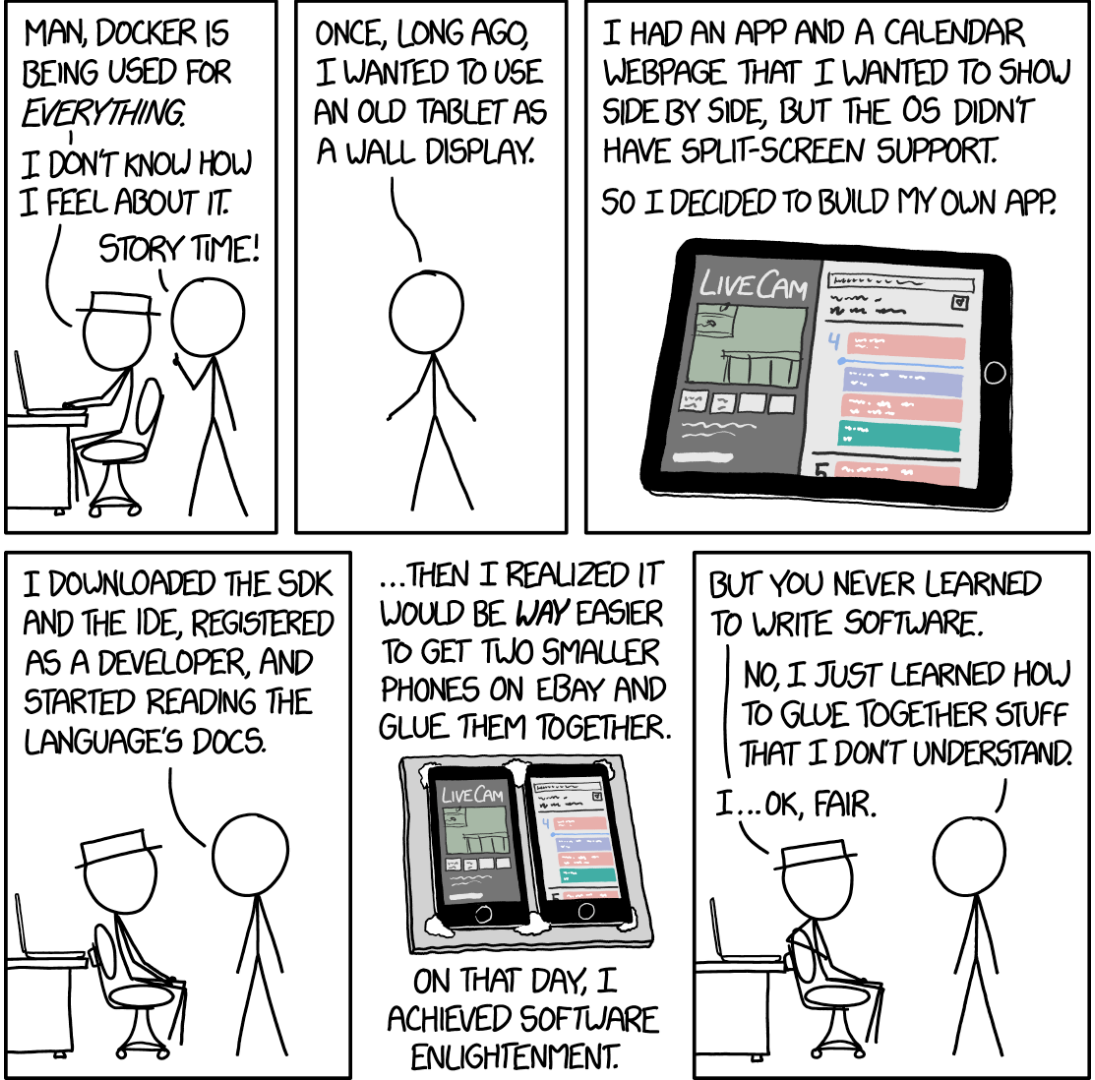
Strategy changes the behaviors of a class on the inside and the decorator(s) change the behavior on the outside

It’s getting cold
Code Example

Docker


docker
Upgrade
shonal107
Containers / Apps
Images
Volumes
Dev Environments
PREVIEW
Search...
Sort by
my-redis redis
EXITED (255) PORT: 6379
redis-docker2 redis:latest
EXITED (255)
redis-docker redis:latest
EXITED (255) PORT: 16379
musing_mayer heartexlabs/lab...
EXITED (255) PORT: 8080
sweet_pike heartexlabs/lab...
EXITED (137) PORT: 8080

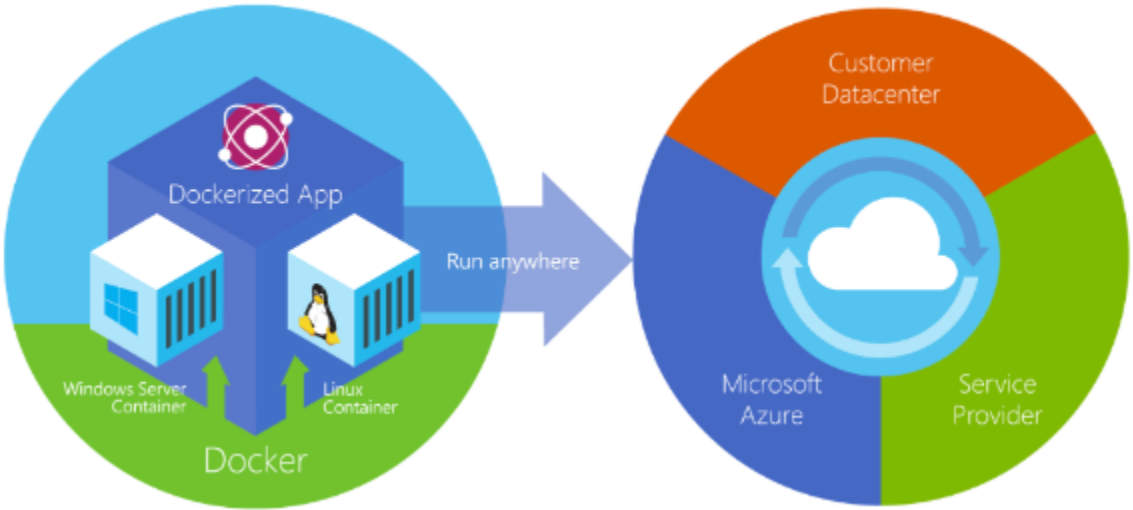
Docker


MAN, DOCKER IS BEING USED FOR EVERYTHING.
I DON'T KNOW HOW I FEEL ABOUT IT.
STORY TIME!
ONCE, LONG AGO, I WANTED TO USE AN OLD TABLET AS A WALL DISPLAY.
I HAD AN APP AND A CALENDAR WEBPAGE THAT I WANTED TO SHOW SIDE BY SIDE, BUT THE OS DIDN'T HAVE SPLIT-SCREEN SUPPORT.
SO I DECIDED TO BUILD MY OWN APP.
LIVECAM
I DOWNLOADED THE SDK AND THE IDE, REGISTERED AS A DEVELOPER, AND STARTED READING THE LANGUAGE'S DOCS.
...THEN I REALIZED IT WOULD BE WAY EASIER TO GET TWO SMALLER PHONES ON EBAY AND GLUE THEM TOGETHER.
LIVECAM
ON THAT DAY, I ACHIEVED SOFTWARE ENLIGHTENMENT.
BUT YOU NEVER LEARNED TO WRITE SOFTWARE.
NO, I JUST LEARNED HOW TO GLUE TOGETHER STUFF THAT I DON'T UNDERSTAND.
I...OK, FAIR.

Docker

this is a joke

Docker


Dockerized App
Windows Server Container
Linux Container
Docker
Run anywhere
Customer Datacenter
Microsoft Azure
Service Provider

Docker

Software that lets you work with containers.

Build, deploy, run, update, and manage containers.

What’s a container? Executable component that combines source code with OS libraries and dependencies required to run the source code.

If you like cloud-native development and hybrid multicloud environments, go read and learn about Docker.

Docker

Better than a VM (virtual machine)

- lightweight: don’t have an entire OS instance, just the OS processes necessary to execute the source code

- Increased productivity – better for Agile and DevOps because they’re ideal for CI/CD because they’re faster and easier to deploy than VMs

- Greater resource efficiency – save money on your cloud spending

Docker Pros

Improved and seamless container portability

Lightweight and granular updates

Automated container creation

Container versioning

Container reuse

Shared container libraries

Docker Terms

DockerFile – list of instructions for Docker to run in order to build the image

Docker Images – executable source code, tools, libraries, and dependencies that runs to make the container. When you run an image, it becomes one instance of the container

Docker containers - Docker containers are the live, running instances of Docker images meaning users can interact with them

Docker Hub – GitHub for Docker

Docker Desktop – the app that I use for Docker stuff, also links to Docker Hub