CORTEX
BY PALO ALTO NETWORKS

Concurrency at Work

Sree Gopinath
Chris Collins
Pamela Toman

August 25, 2021

paloalto NETWORKS

What are we here for?

We're going to delve into key ideas that don't always get covered in school but matter in professional contexts.

1. **Sree** will share how Cortex Xpanse performs "port scanning" to know what is actually on the internet; this was the seed of Expanse-the-startup

2. **Chris** will share how Cortex Xpanse uses "functional programming" to write data processing code that parallelizes cleanly (modern MapReduce)

3. **Pamela** will share how Cortex Xpanse uses "microservices" to make teams & services independent of each other for ML

All of this relates directly to what you've learned this quarter.

It will be useful for you in becoming (& being perceived as) a Real Software Engineer™!

Cortex Xpanse is a leader in attack surface management

Attack Surface

Attack surface management is about accidental vulnerabilities

Bad actors are **constantly** checking for vulnerabilities

- Weaknesses are easily detected
- Weaknesses are easily exploited
- Everyone has weaknesses

Chris Krebs @C_C_Krebs · 19h

This is the real deal. If your organization runs an OWA server exposed to the internet, assume compromise between 02/26-03/03. Check for 8 character aspx files in C:\\inetpub\wwwroot\aspnet_client\system_web\. If you get a hit on that search, you're now in incident response mode.

Jake Sullivan @JakeSullivan46 · 20h

We are closely tracking Microsoft's emergency patch for previously unknown vulnerabilities in Exchange Server software and reports of potential compromises of U.S. think tanks and defense industrial base entities. We encourage network owners to patch ASAP: msrc-blog.microsoft.com/2021/03/02/mul...

Show this thread

26 1.3K 2.5K

Cortex Xpanse discovers what is on a network that...

- **The company didn't even know about**
- **The company knew about but had accidentally misconfigured**

Unmanaged and misconfigured assets compromise security.

What does the computer system need to do?

To find and surface crackable web services, we need to....

- Monitor what's on "the internet"
- Figure out which of those things are "yours"
- Assess "dangerousness"
- Process & display all that

This needs to happen at scale and stay current

It's a hard problem (solving it is worth at least $800 million)

Let’s design a system for attack surface management

Collect facts about the internet

Anyone who buys a domain/certificate/IP must provide contact info:

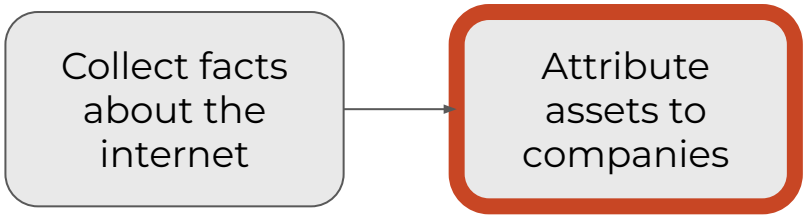
Domain	stanford.edu
Collected	Mar 28, 2021
Created	
Updated	
Expiry	
Registrar Name	
Registrar Whois	
Name Servers	AVALLONE.STANFORD.EDU, ATALANTE.STANFORD.EDU, NS6.DNSMADEEASY.COM, ARGUS.STANFORD.EDU, NS7.DNSMADEEASY.COM, NS5.DNSMADEEASY.COM

Name	Stanford University|The Board of Trustees of the Leland Stanford Junior University
Org	
Street	241 Panama Street, Pine Hall, Room 115
City	Stanford
Province	CA
Postal Code	94305-4122
Country	UNITED STATES
Phone	
Fax	
Email	

Name	Domain Admin|Stanford University
Org	
Street	241 Panama Street Pine Hall, Room 115
City	Stanford
Province	CA
Postal Code	94305-4122
Country	UNITED STATES
Phone	16507234328
Fax	
Email	sunet-admin@stanford.edu

(We know more about assets/companies than just their registration info)

Let’s design a system for attack surface management



domain
cmcrequest.com

IP address
34.107.151.202

certificate
MIIGbDCCBVSg...

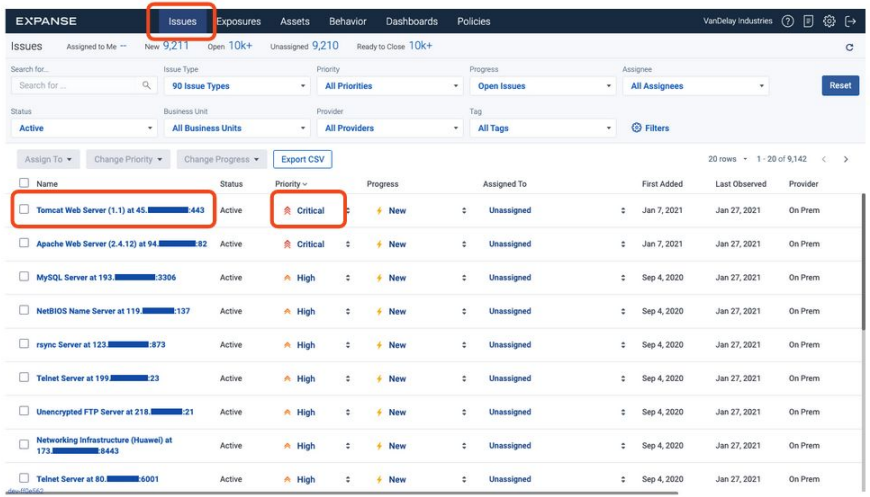
?

CVS Health

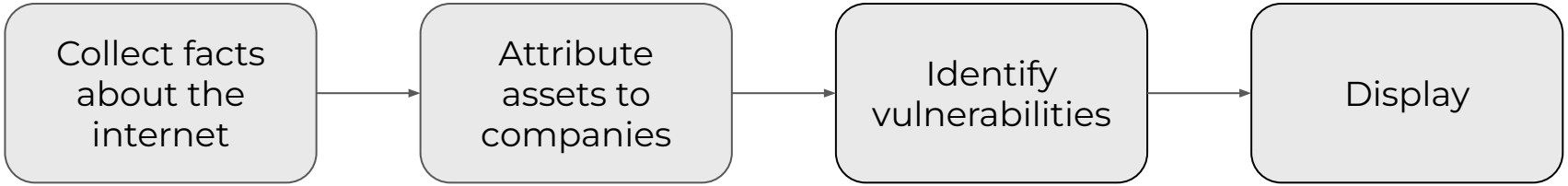
accenture

Let’s design a system for attack surface management

Collect facts about the internet → Attribute assets to companies → Identify vulnerabilities → Display



Engineering teams support each subsystem



We're focusing on the aspects most relevant to CS 110.

Sree
Port scanning

Pamela
Microservices

Chris: Data processing

CORTEX
BY PALO ALTO NETWORKS

<snip 57 slides>

The other sections belong to other people. I'm not sharing them.

paloalto NETWORKS

CORTEX
BY PALO ALTO NETWORKS

Microservices: The whats & whys

Pamela Toman
Machine Learning Engineering

August 2021

paloalto NETWORKS

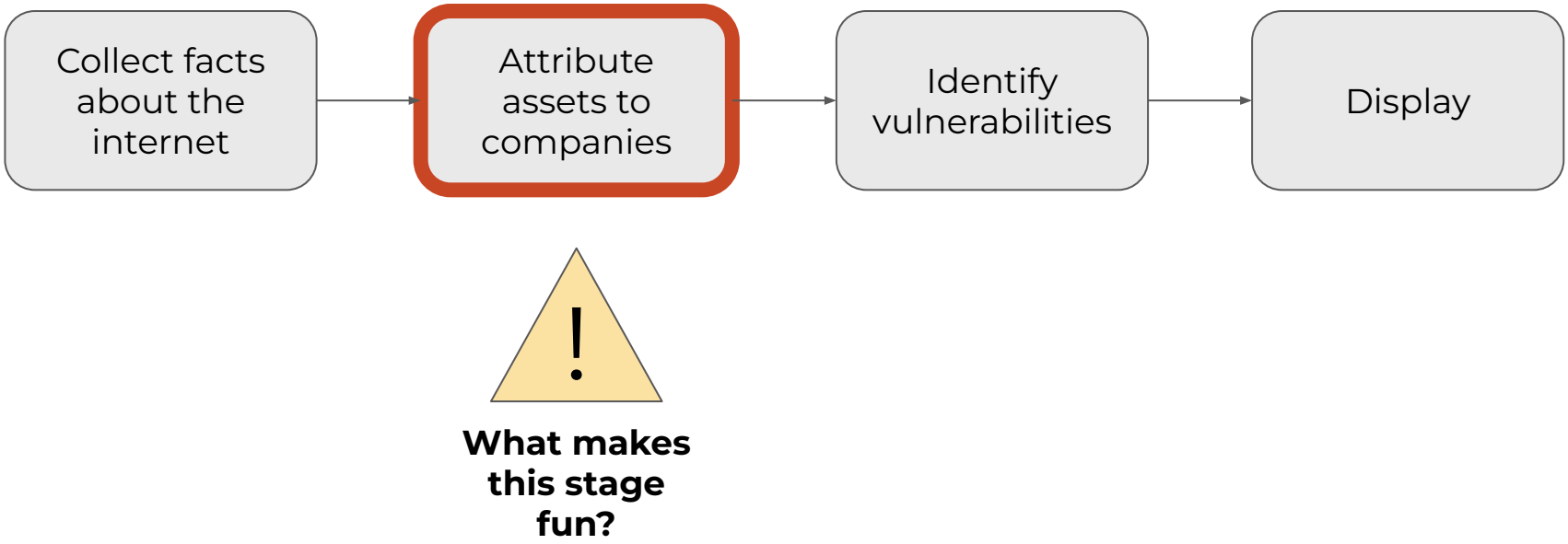
Key takeaways

- It's common to deploy machine learning models as "microservices"
  - Customers pay Cortex Xpanse to find their unknown-and-crackable web services quickly
  - Machine learning helps us identify "what services belong to whom"
- Microservices *decouple* your application into subcomponents that scale *independently*
- CS 110's core design concepts extend to networked environments:
  - Each worker does a single thing
  - Pools of workers share a single point of entry
  - Communication happens through a request/response model
- Microservices are more work but sometimes very useful

Let’s design a system for asset attribution...

(like an interview)

Let’s design a system for attack surface management



Rephrase the goal:
For each asset on the internet, who does it belong to?

Knowing what is exposed on your full attack surface is non-trivial

Identifying what an organization owns is **hard**

The engineering problem:

- Internet-scale
- Balance between speed & money
- Deploying, managing, and monitoring

The machine learning problem:

- Asset change & churn
- Shadow IT
- Mergers & acquisitions, divestitures, ...

Cortex Xpanse generally finds 10-30% more assets than customers were tracking

Global internet data (PBs!)

Organization-specific data

We want a *ton* of context for the ML model (columns). And we're aware of a *ton* of assets on the internet (rows). Our data are expensive to process!

What do we do?

Let’s reframe as 2 problems: “filtering” separate from “prediction”

Filter likely asset-company pairs

Predict which pairs are truly owned

It’s common that MLE uses different core tech from non-MLE

Filter likely asset-company pairs

Big Data Engineering
Java
Beam/Dataflow

Machine Learning Engineering
Python
Flask/gunicorn

Predict which pairs are truly owned

How can we build a system that....

- Uses a **different language & team** for each subcomponent?

AND

- **Scales** to the size of the internet?

AND

- **Costs** no more resources than necessary?

Microservices to the rescue!

A powerful concept in concurrency:
Total independence of parts.

Independence is good for:
(1) team ownership & (2) appropriate scaling

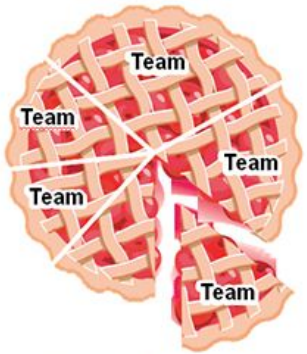
(But what does that even tangibly *mean*? How does it manifest in Engineering-land?)

Microservices decouple subcomponents via network APIs

Microservices apply two core ideas:

- **Decoupling**: no shared knowledge between units
  (always good! like pipeable shell commands, or decomposing code into functions)

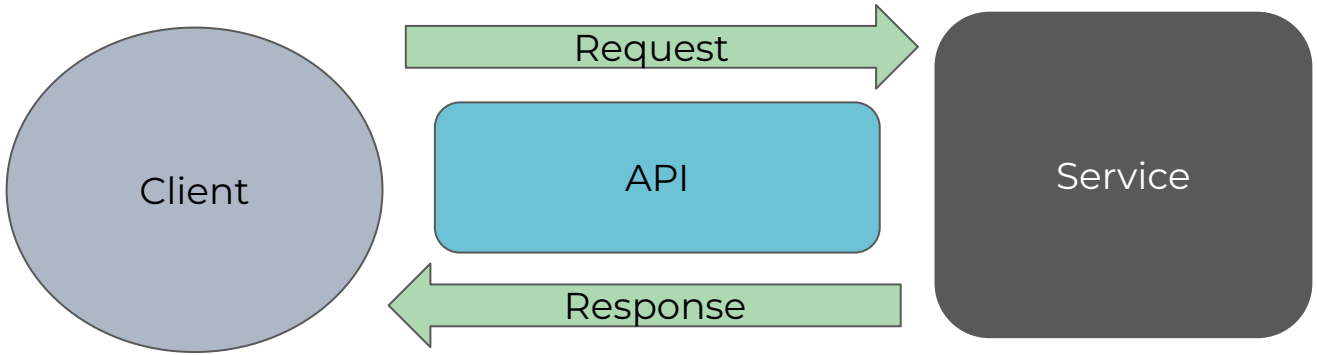
- **Networked communication**: all inter-unit communication happens over web APIs
  (neutral usefulness)



Microservices are usually deployed over the internet

The “which company does this asset belong to” service:

- Listens for connections on the `/predict` endpoint
- Validates the requests
- Executes
- Responds in expected format



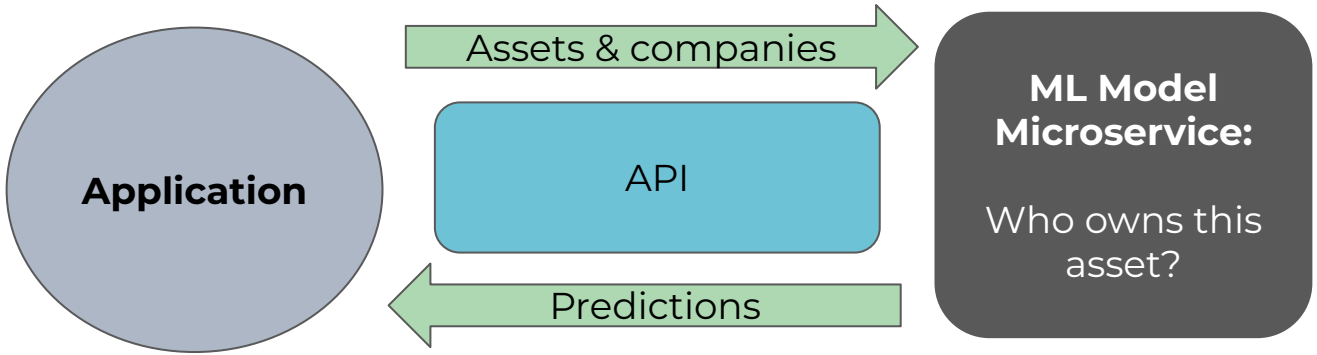
The service is a black box

We can change it however we want (the code, libraries, language, ML features, anything)

Microservices are usually deployed over the internet

The “which company does this asset belong to” service:

- Listens for connections on the `/predict` endpoint
- Validates the requests
- Executes
- Responds in expected format



The service is a black box

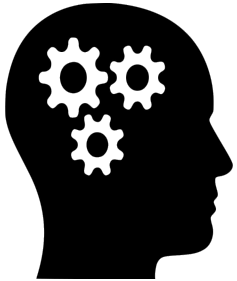
We can change it however we want (the code, libraries, language, ML features, anything)

Even with our service running, it’s not yet useful in production

So far we have...

- Code that predicts ownership of assets
- Ability to send a message to `/predict` and get a response
- A set of assets & companies we want to evaluate

But on its own, that’s still not enough.



We’ve got a service in our dev environment. Now we need to productionize it.

What do we consider?

We need to know about the workload

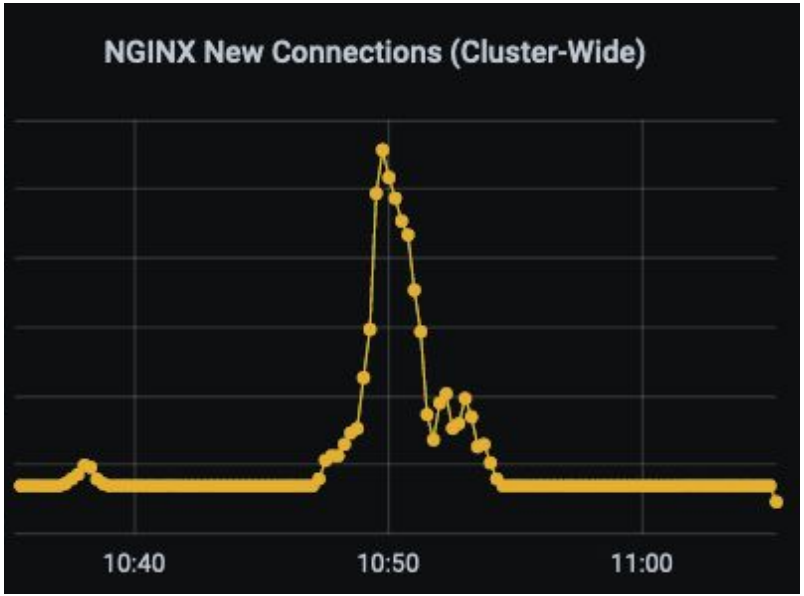
The ML prediction service needs to serve a *lot* of requests:

- Many millions of requests each day
- They come in high-volume batches

The attribution workload is high-volume and spiky

One machine can't handle all the requests.

What do we do?



We duplicate with multiple machines *and* “containerization”

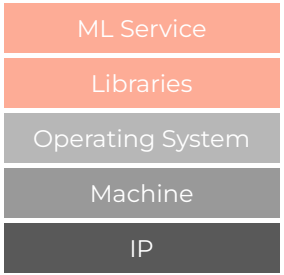
Similar to duplicating processes with `fork...`

We duplicate *services* with “containerization” (e.g., Docker)

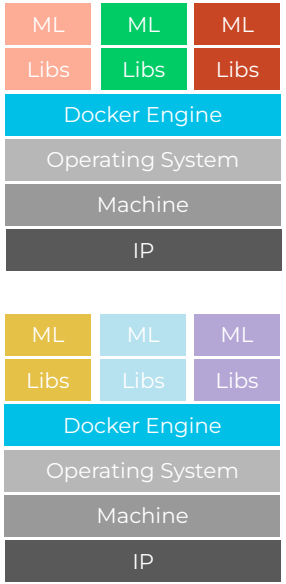
- Each service lives in a very lightweight quasi-virtual machine
- Duplicates (replicas) are entirely identical
- We fit more than 1 container per machine
- It’s very fast to boot up more containers

Now the ML can predict on thousands of requests near-simultaneously and cheaply!

Duplicating one service per machine takes a LOT of resources



Duplication with containers is more efficient



We can create service replicas on the fly. But how do we know *when* to create more?

What do we do?

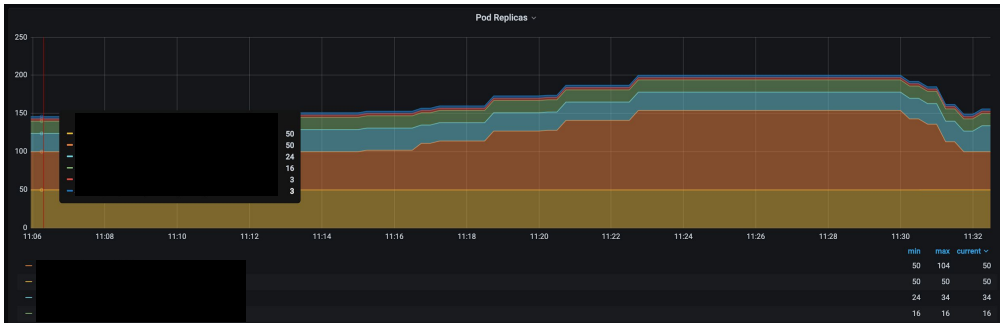
We use orchestration software to manage *how many* replicas

Orchestration software (e.g., Kubernetes) manages the “container lifecycle”:

- When do we need to bring up more replicas?
- When can we kill replicas?

We describe what we want, and let Kubernetes figure out “how”

- We get to ignore how containerized replicas map to physical machines
- We get to ignore whether there are “enough” copies
- The service scales based on *actual* need, independent of other components



We have an ever-changing number of replicas. How does a client actually *find* a copy of the service?

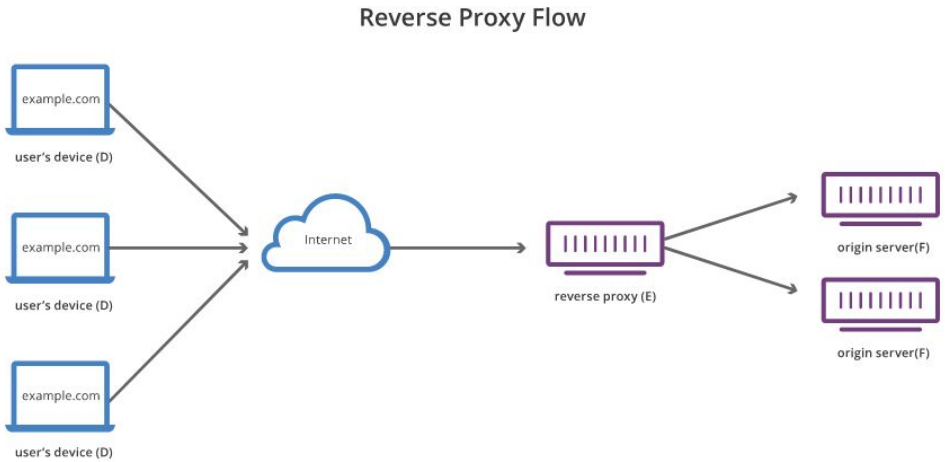
What do we do?

We need a single point of entry

If we have an arbitrary number of replicas, **how does the backend client know which copy of the ML prediction service to talk to?**

We introduce a layer of abstraction:

- A “reverse proxy” sits in front of all the duplicates (e.g., nginx)
- The client knows about 1 location: the proxy
- The proxy knows about N locations: the service replicas that run the workload



This is similar to a threadpool!

(...if the threadpool had another software layer that dynamically adjusted the number of threads)

To recap....

Microservices decouple complicated code (like ML models) from the rest of the system:

- The “services” can be **black boxes** to everyone else
- They come **bundled** with everything they need to run
- The containerized bundle can get **duplicated** if the load goes up
- We use a reverse proxy as a **single point of entry**

Microservices are particularly useful if you want **independence** of parts
(multiple teams, multiple languages, independent scaling of subcomponents)

There are **costs** to a microservice architecture:

- More dependencies (like the network) mean more possible points of failure
- Network communications are slow
- You’ve got to write & maintain more code
- You’ve got to be comfortable with a larger number of technologies

Key takeaways

- It's common to deploy machine learning models as "microservices"
  - Customers pay Cortex Xpanse to find their unknown-and-crackable web services quickly
  - Machine learning helps us identify "what services belong to whom"
- Microservices *decouple* your application into subcomponents that scale *independently*
- CS 110's core design concepts extend to networked environments:
  - Each worker does a single thing
  - Pools of workers share a single point of entry
  - Communication happens through a request/response model
- Microservices are more work but sometimes very useful

CORTEX
BY PALO ALTO NETWORKS

Thank you

paloaltonetworks.com

paloalto NETWORKS