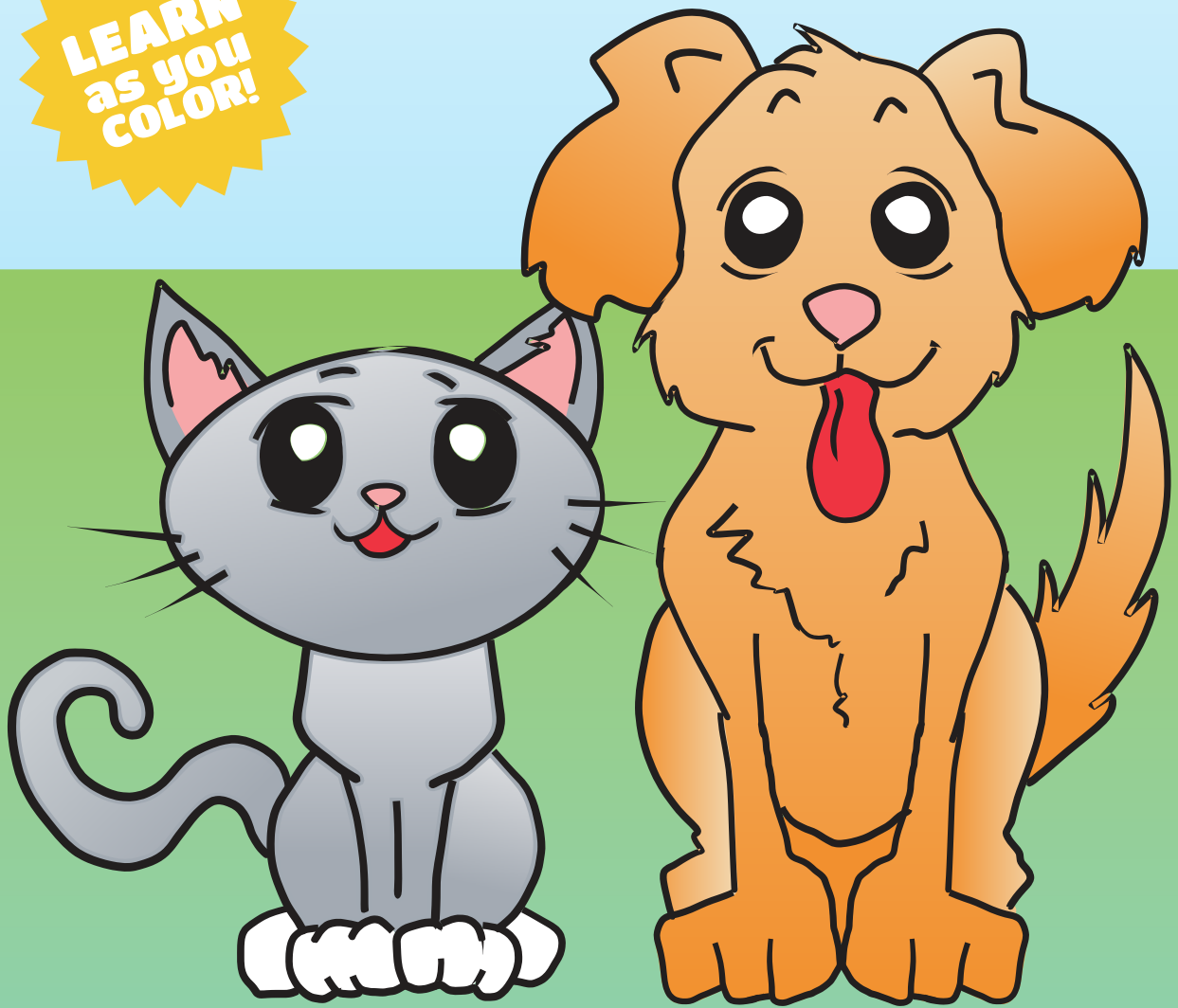
Sponsored by Red Hat

the SELINUX COLORING BOOK

"It's raining cats and dogs!"

LEARN as you COLOR!



written by DAN WALSH

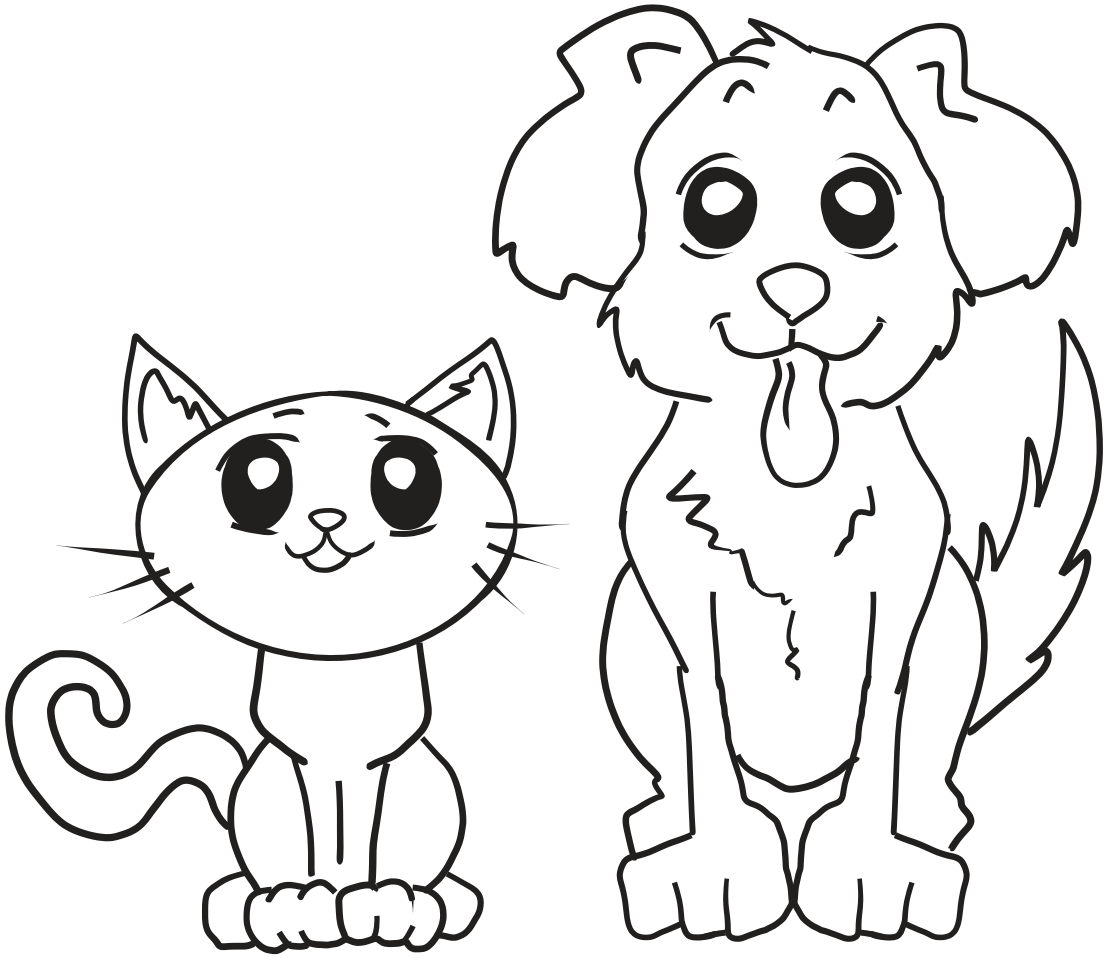
illustrated by MÁIRÍN DUFFY

TYPE ENFORCEMENT

PROCESS TYPES

The SELinux primary model of enforcement is called type enforcement. Basically, this means we define the label on a process based on its type, and the label on a file system object based on its type.

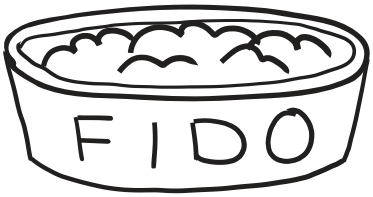
Imagine a system where we define types on objects like cats and dogs. A cat and dog are process types.



OBJECT TYPES

We have a class of objects that they want to interact with which we call food. And I want to add types to the food, cat_chow and dog_chow.

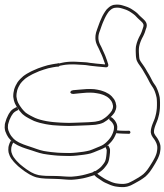
CAT_CHOW



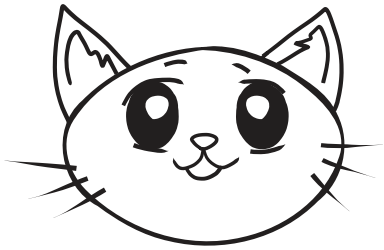
DOG_CHOW

POLICY RULES

As a policy writer, I would say that a dog has permission to eat dog_chow food and a cat has permission to eat cat_chow food. In SELinux we would write this rule in policy, as shown below:

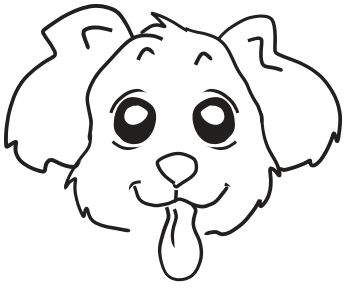


ALLOW



CAT

ALLOW

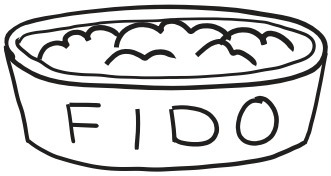


DOG

CAT_CHOW:FOOD

+

EAT

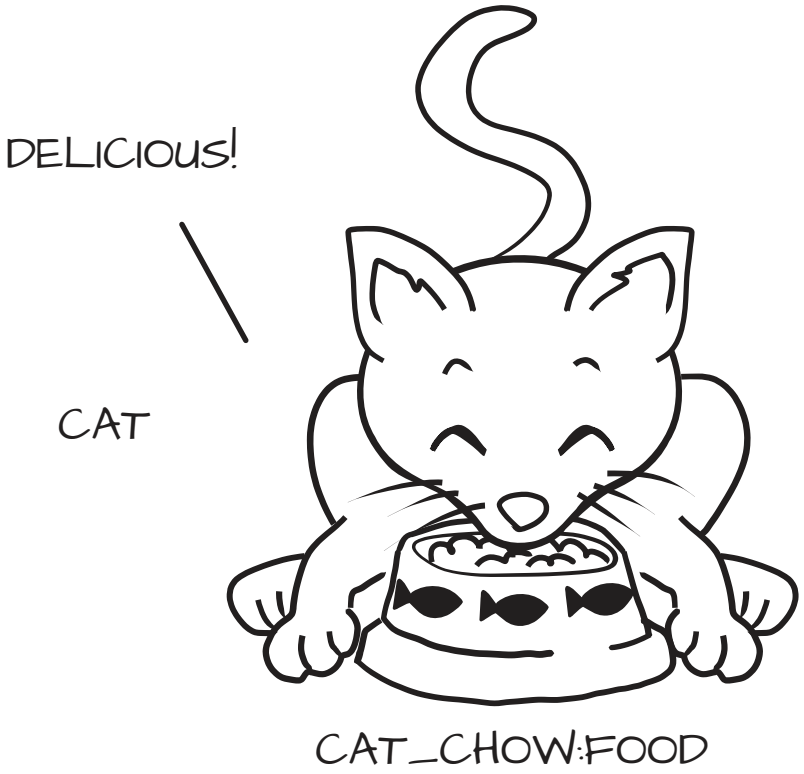


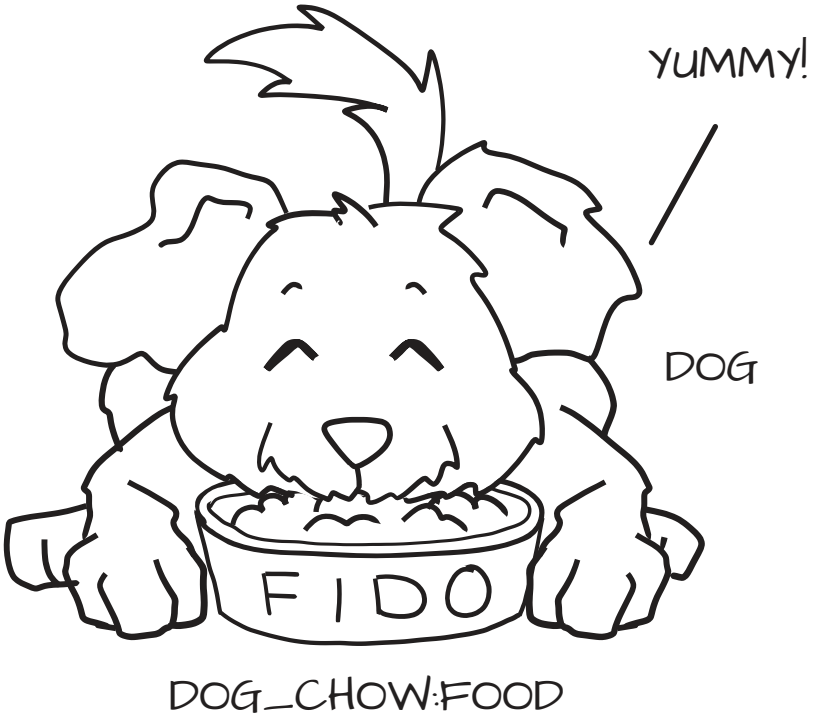
DOG_CHOW:FOOD

+

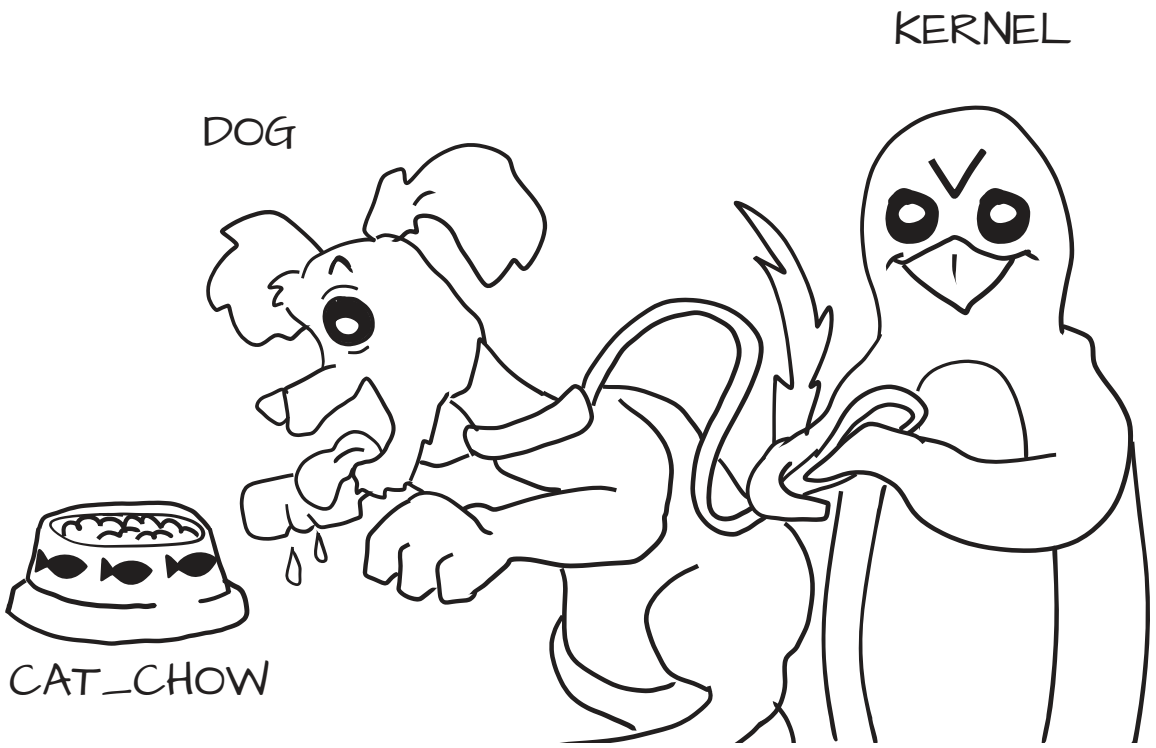
EAT

With these rules the kernel would allow the cat process to eat food labeled cat_chow and the dog to eat food labeled dog_chow.

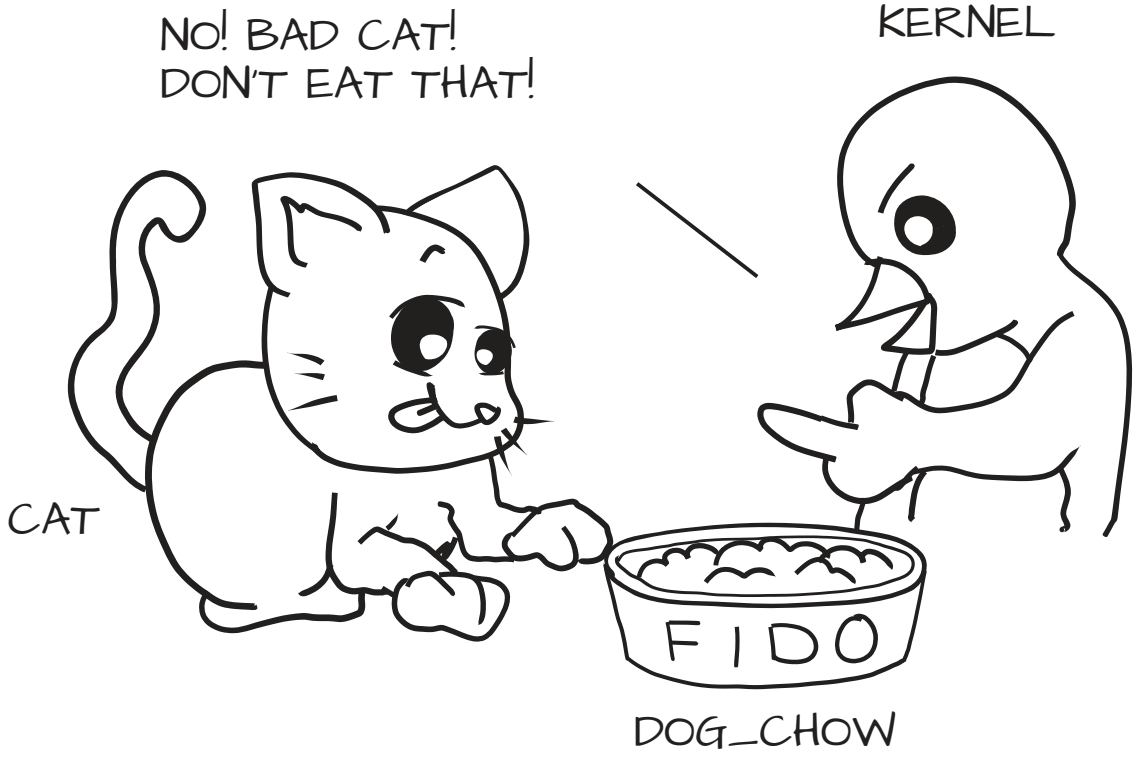




But in an SELinux system, everything is denied by default. This means that if the dog process tried to eat the cat_chow, the kernel would prevent it.

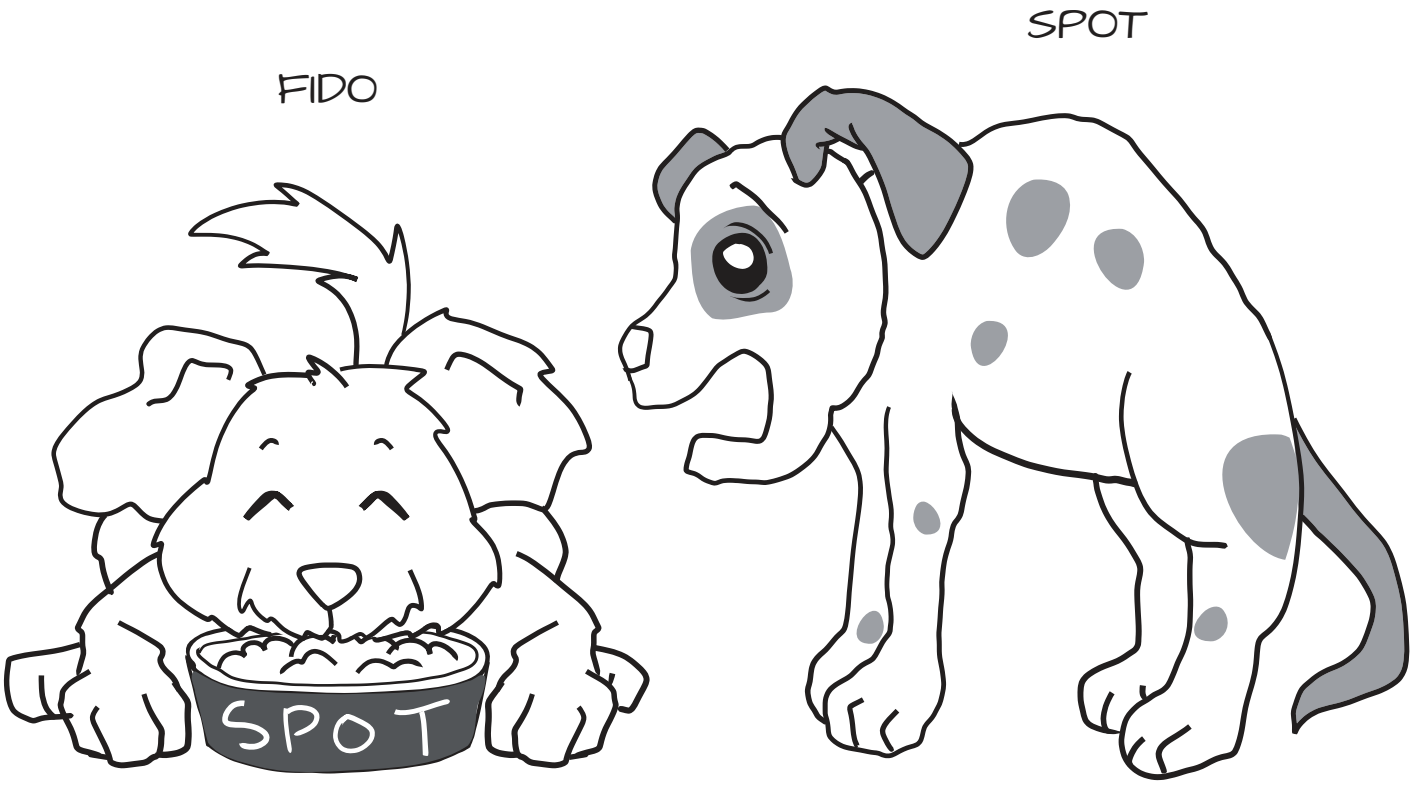


Likewise, cats would not be allowed to touch dog food.



MCS ENFORCEMENT

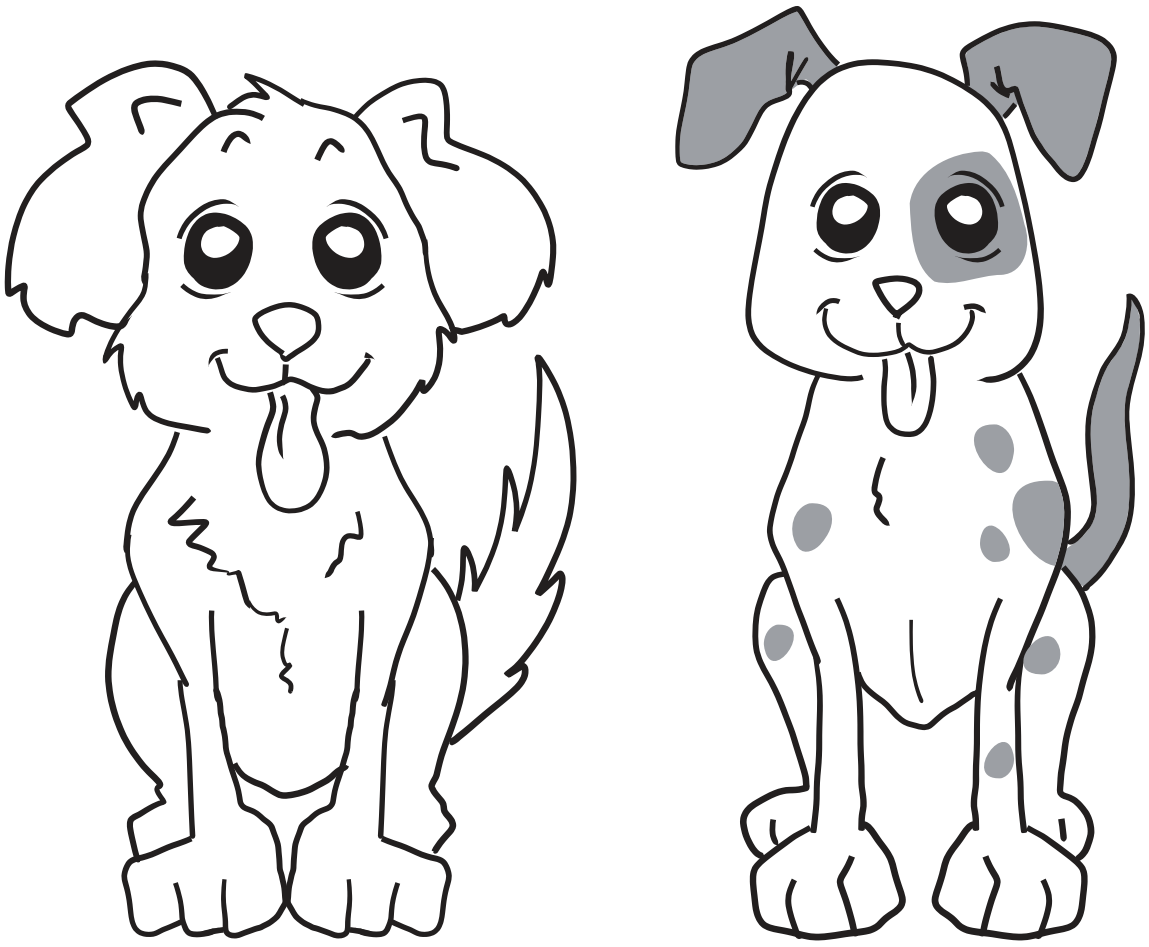
We've typed the dog process and cat process, but what happens if you have multiple dog processes: Fido and Spot? You want to stop Fido from eating Spot's dog_chow.



One solution would be to create lots of new types, like Fido_dog and Fido_dog_chow. But, this will quickly become unruly because all dogs have pretty much the same permissions.

To handle this we developed a new form of enforcement, which we call Multi Category Security (MCS). In MCS, we add another section of the label which we can apply to the dog process and to the dog_chow food. Now we label the dog process as dog:random1 (Fido) and dog:random2 (Spot).

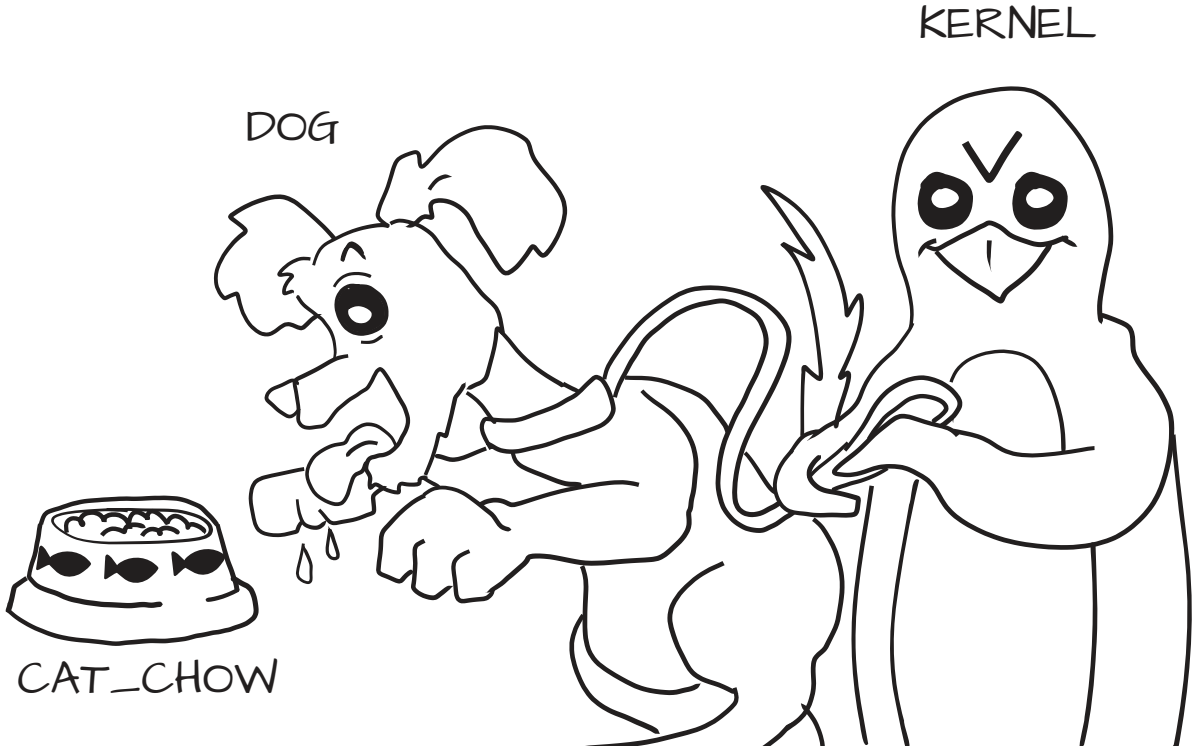
We label the dog chow as dog_chow:random1 (Fido) and dog_chow:random2 (Spot).



MCS rules say that if the type enforcement rules are OK and the random MCS labels match exactly, then the access is allowed. If not, it is denied.

TYPE ENFORCEMENT

Fido (dog:random1) trying to eat cat_chow:food is denied by type enforcement.



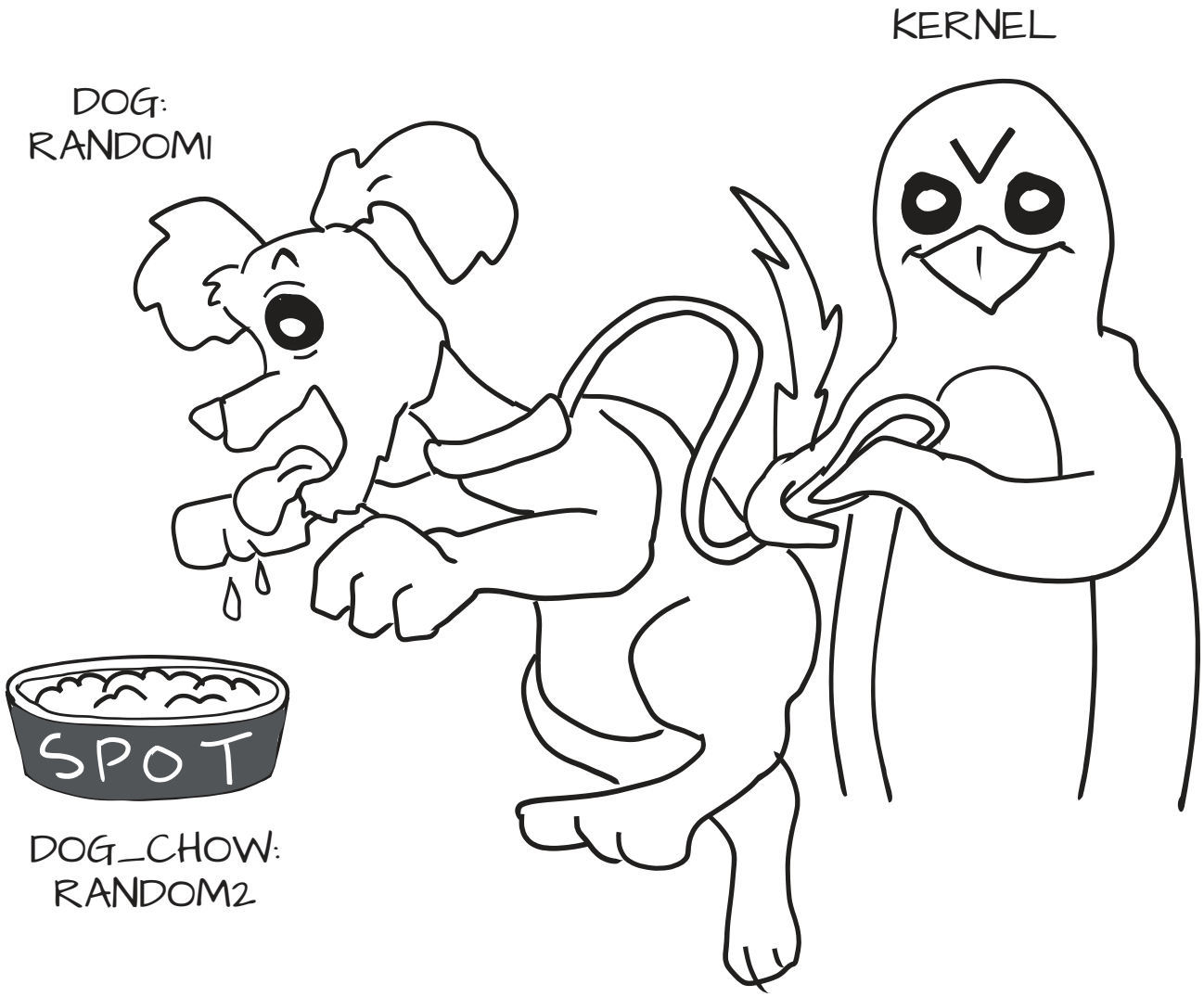
Fido (dog:random1) is allowed to eat dog_chow:random1.

DOG:RANDOMI

DOG_CHOW:
RANDOMI

MCS ENFORCEMENT

Fido (dog:random1) denied to eat spot's (dog_chow:random2) food.

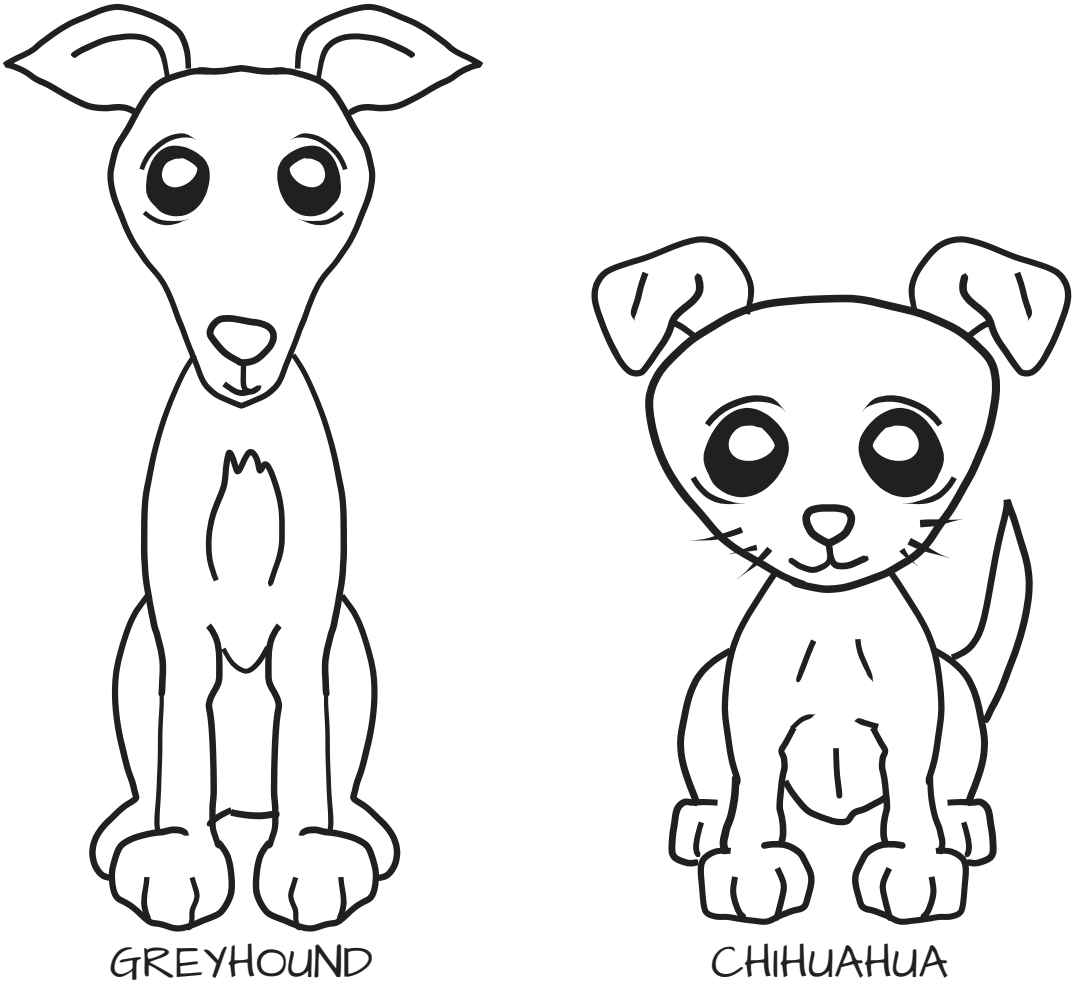


MLS ENFORCEMENT

Another form of SELinux enforcement, used much less frequently, is called Multi Level Security (MLS); it was developed back in the 60s and is used mainly in trusted operating systems like Trusted Solaris.

The main idea is to control processes based on the level of the data they will be using: a secret process can not read top-secret data.

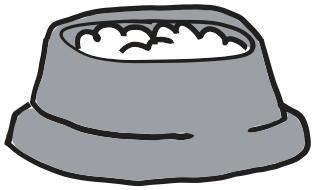
Instead of talking about different dogs, we now look at different breeds. We might have a Greyhound and a Chihuahua:



We might want to allow the Greyhound to eat any dog food, but a Chihuahua could choke if it tried to eat Greyhound dog food.

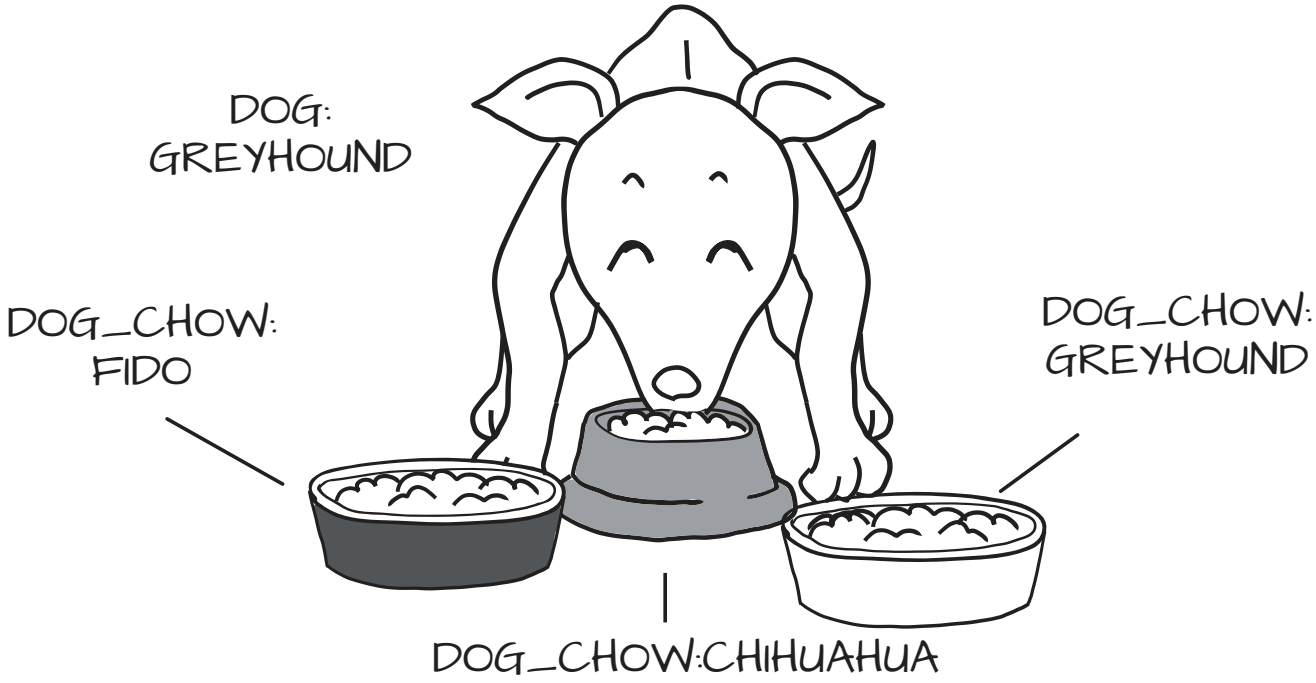
We want to label the Greyhound as dog:Greyhound and his dog food as dog_chow:Greyhound, and label the Chihuahua as dog:Chihuahua and his food as dog_chow:Chihuahua.

DOG_CHOW:
GREYHOUND



DOG_CHOW:
CHIHUAHUA

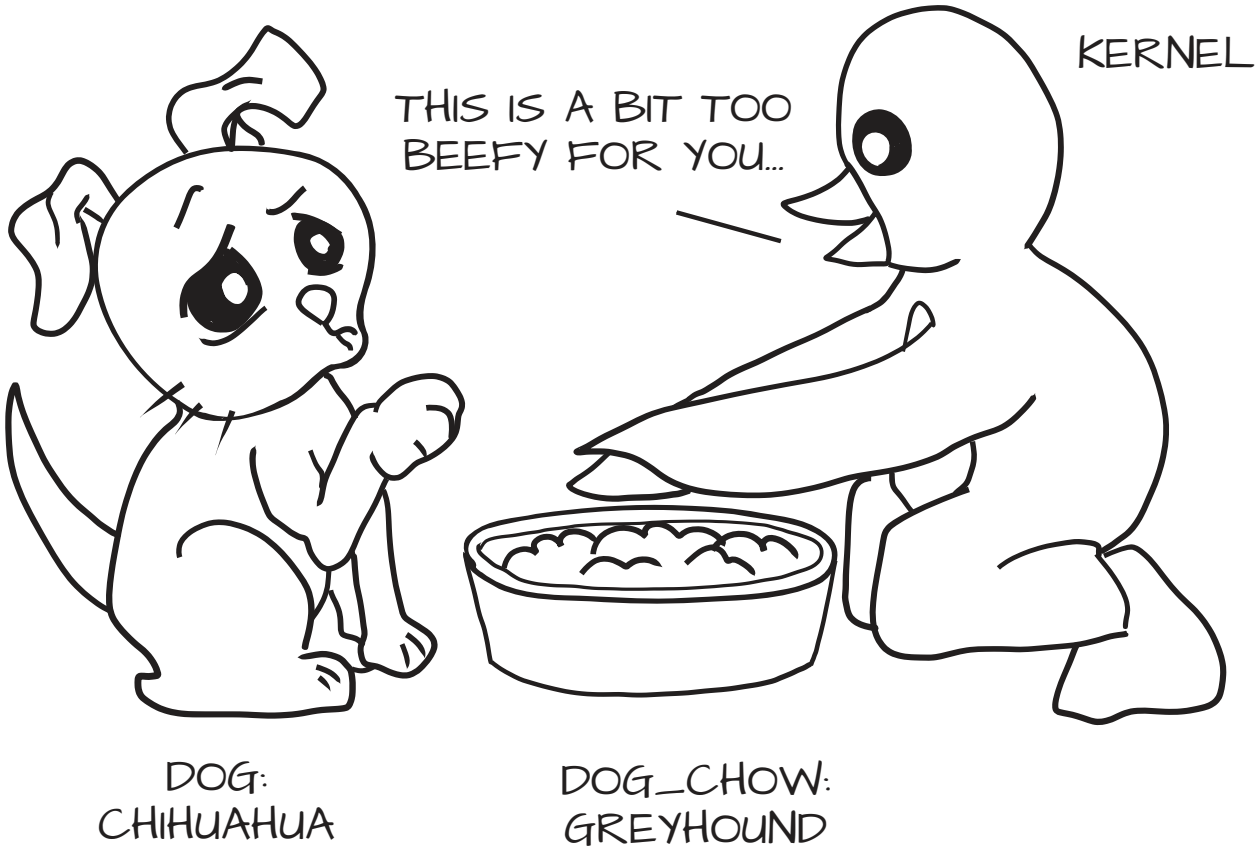
With the MLS policy, we would have the MLS Greyhound label dominate the Chihuahua label. This means dog:Greyhound is allowed to eat
dog_chow:Greyhound and dog_chow:Chihuahua.



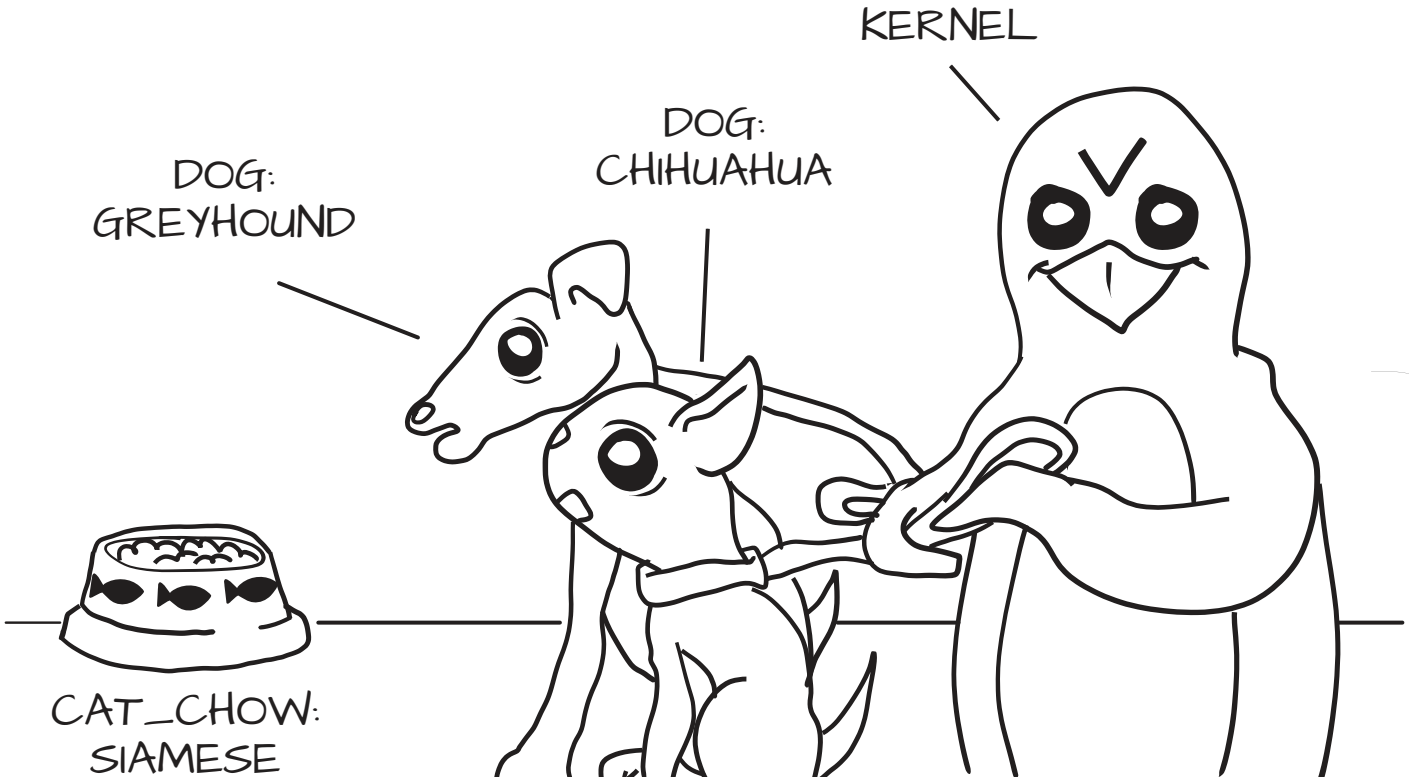
DOG:
CHIHUAHUA

DOG_CHOW:
CHIHUAHUA

But dog:Chihuahua is not allowed to eat dog_chow:Greyhound.



Of course, dog:Greyhound and dog:Chihuahua are still prevented from eating cat_chow:Siamese by type enforcement, even if the MLS type Greyhound dominates Siamese.



Learn more at redhat.com:

http://red.ht/security

Sponsored by Red Hat

the CONTAINER COLORING BOOK

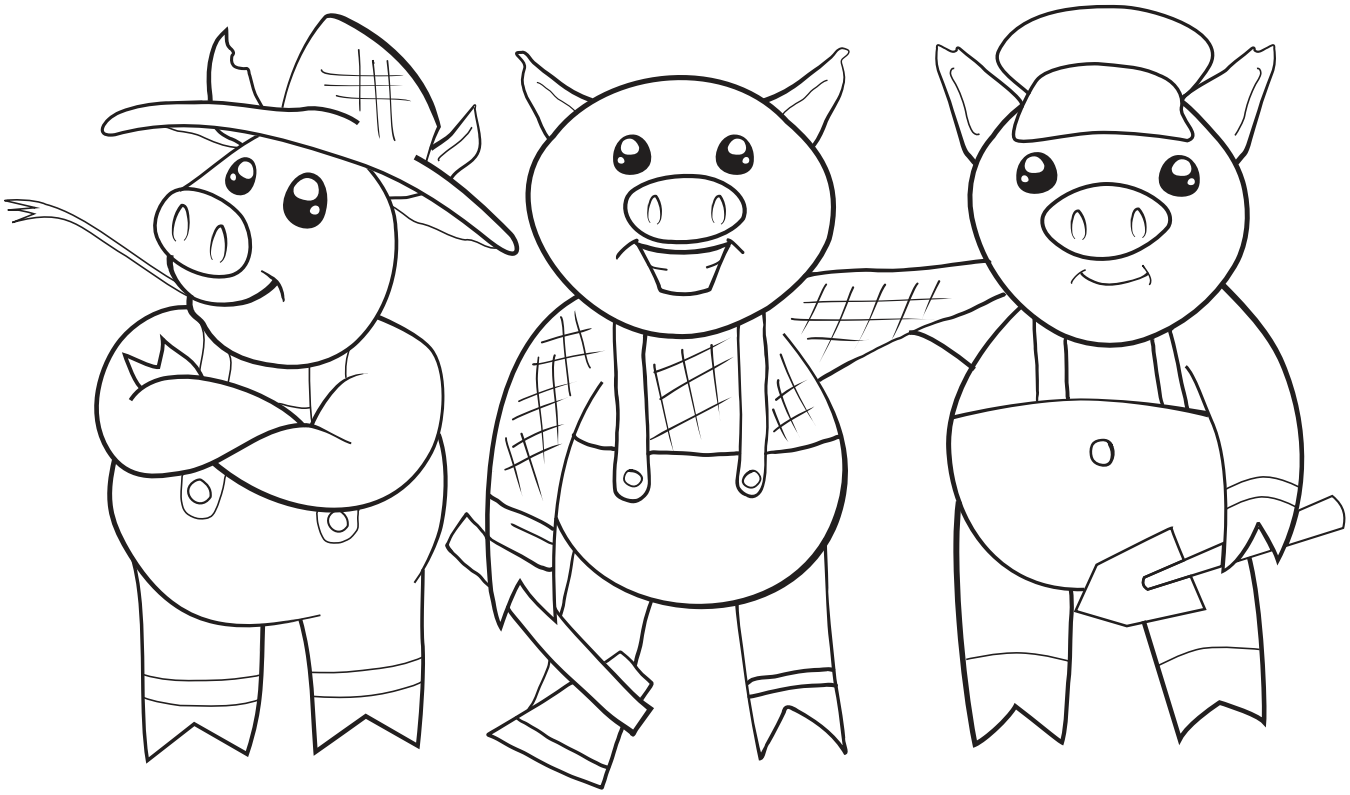
"Who's afraid of the big bad wolf?"

written by DAN WALSH

illustrated by MÁIRÍN DUFFY

INTRODUCTION

Once upon a time, there were three little pigs. They each needed a place to live.

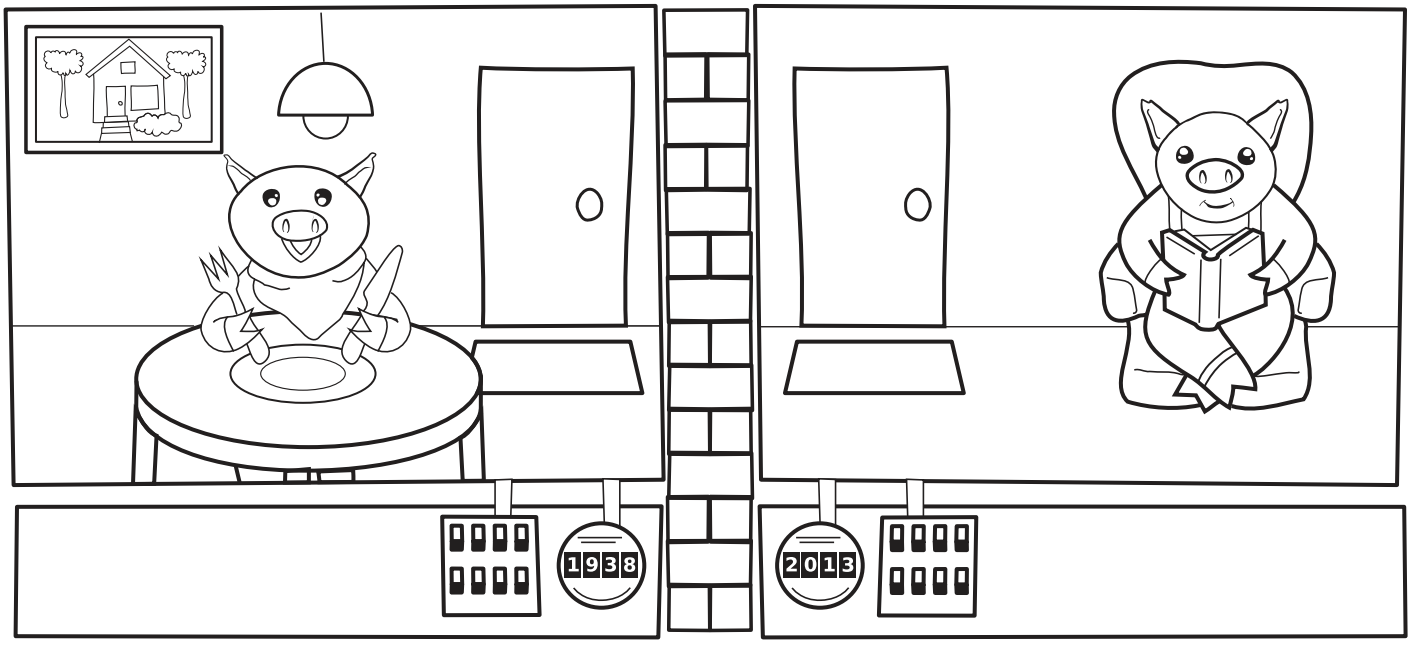


There's a lot of different types of places to choose from...

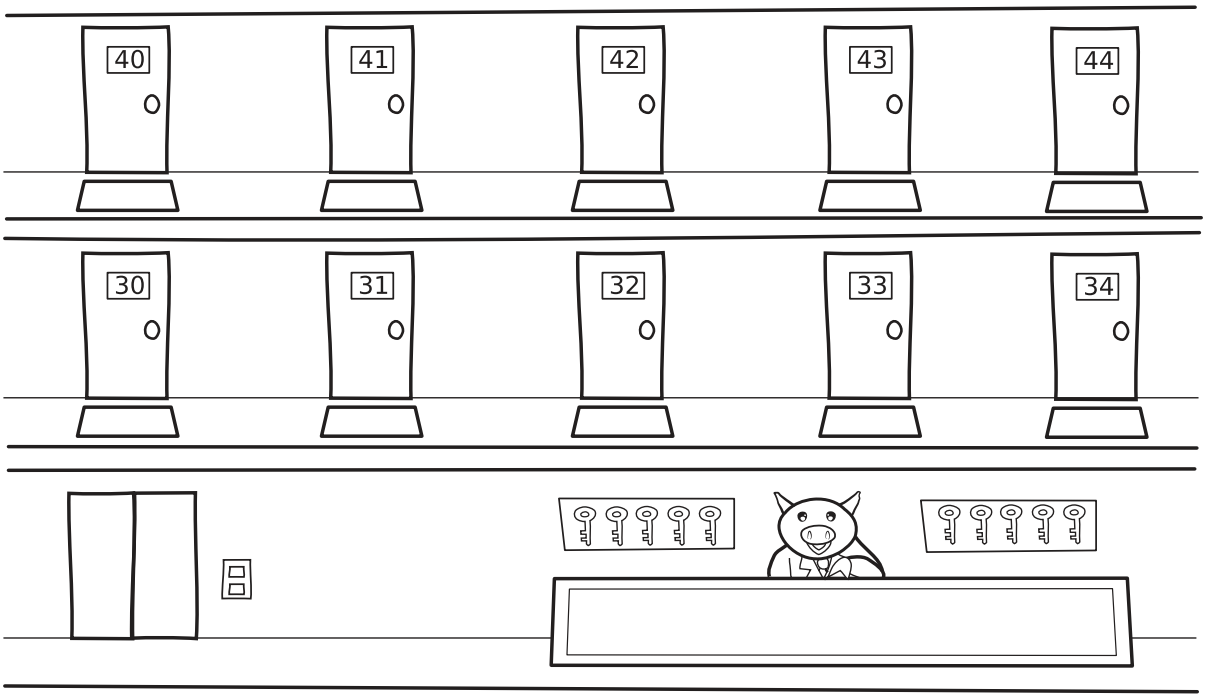
HOUSE DUPLEX APARTMENT HOSTEL PARK

If a piggy was an application.... living in a house (physical machine) would be the most secure. If one house is broken into, the other houses remain secure. A separate house per piggy means a lot more home maintenance, though!

A piggy living in a duplex is like an application with multiple services deployed to multiple VMs on the same physical machine. While the structure is shared, the entry points are not. If one home is compromised, breaking in to the other VMs involves breaking through the hypervisor, sVirt, and the host kernel. However, you still have the costs of maintaining multiple OSes, with loss of speed and a limited ability to share resources.

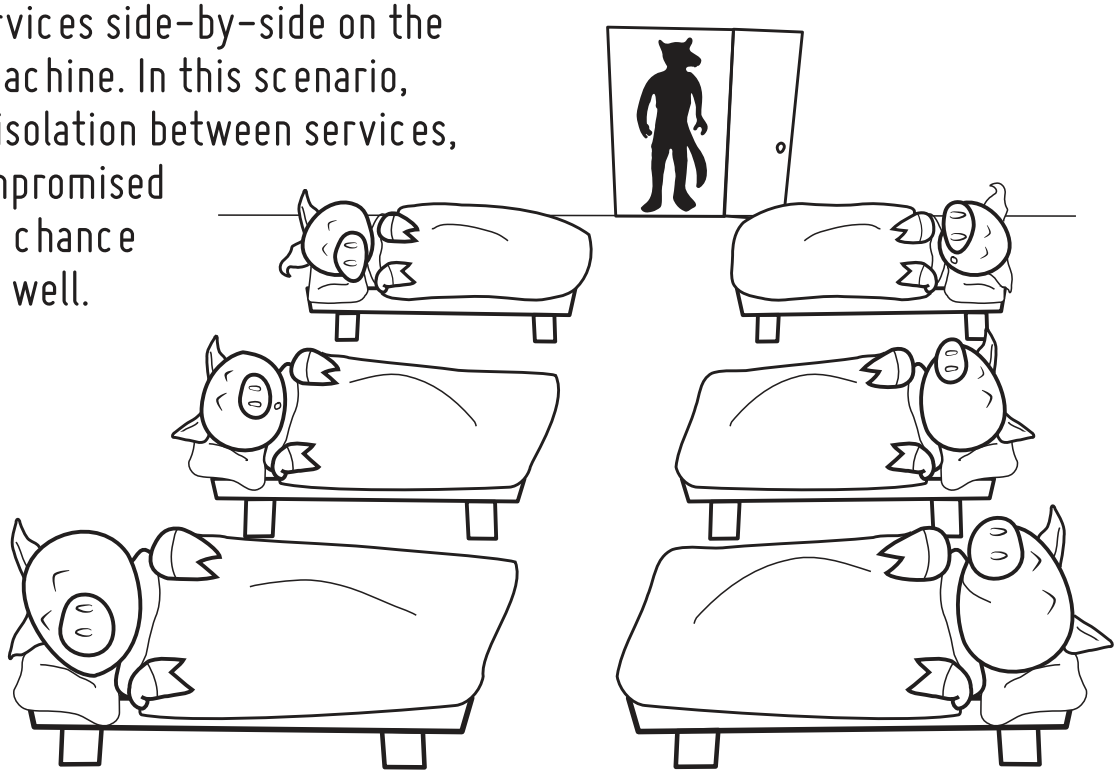


Piggies living in an apartment building are like applications running in containers. You get excellent sharing of services, lower cost of maintainence and decent separation. One problem, though, is that if the front desk were compromised, then all of the apartments would be compromised. This is similar to a container environment where, if the kernel were compromised, all of the containers would be as well.

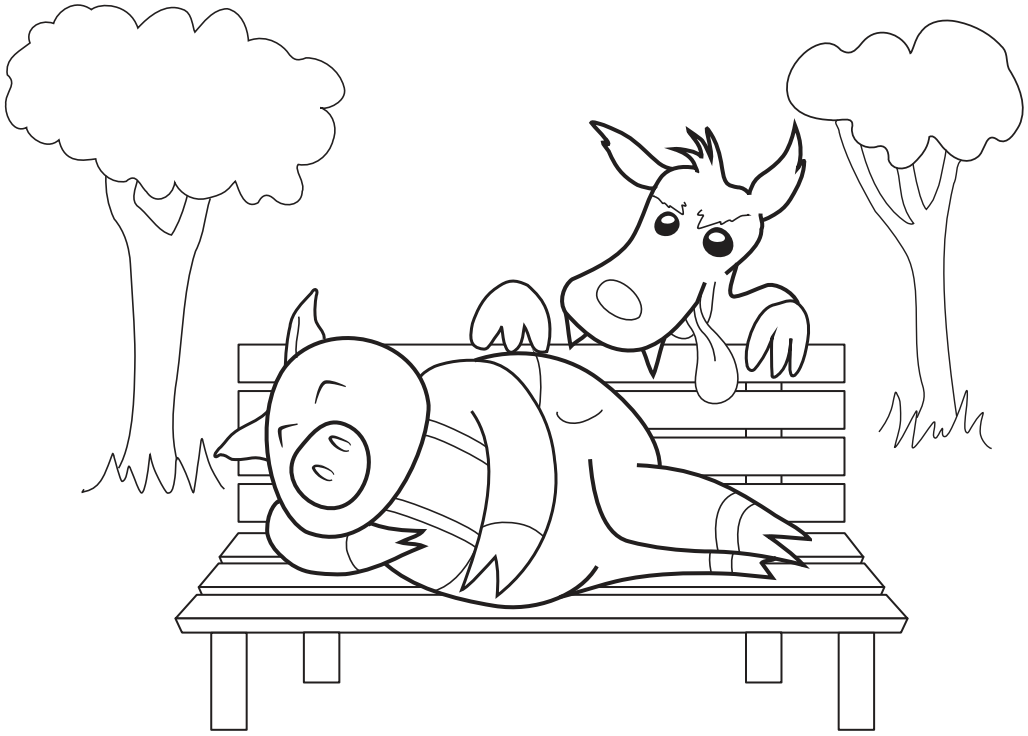


Piggies living in a hostel are like running an application's services side-by-side on the same physical machine. In this scenario, there is limited isolation between services, but if one is compromised there is a strong chance others will be as well.

Of course, if you're running with SELinux, you'll have better isolation.

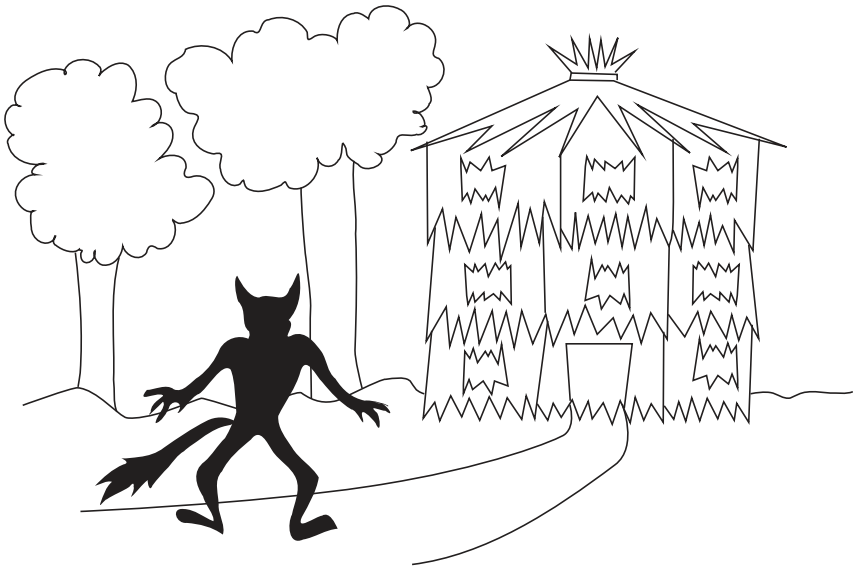


If they are up for living on the edge – as folks who run their apps on systems running setenforce 0 are – the piggy could consider sleeping in the park. We don't need to tell you how risky this is.



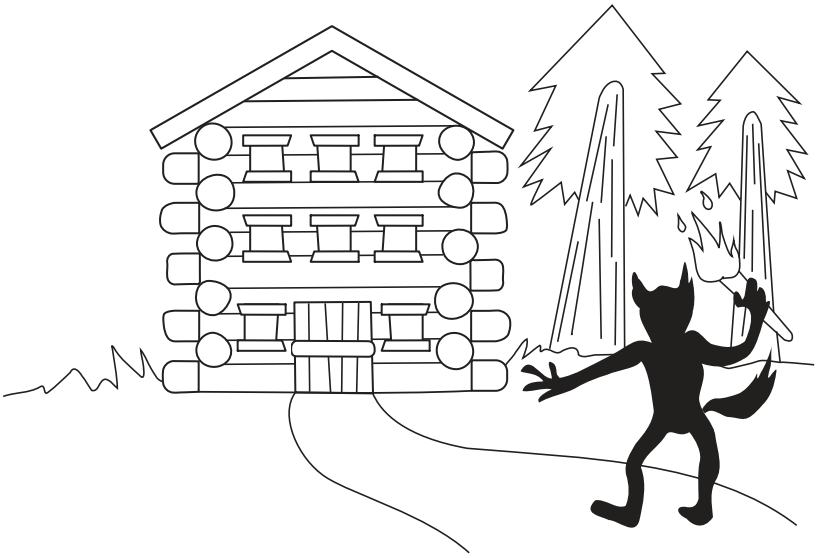
Containers, as represented by the apartment building, seem like a good middle ground. The apartment building offers better security than services sharing the same host, with more flexibility on content. Apartments provide better sharing of resources, startup speeds, and the cost of maintenance is lower than duplexes (VMs). Let's explore life at the apartment building in greater detail.

When choosing an apartment building to live in or a host platform to run your containers, construction quality is a top concern.



Running containers on a do-it-yourself platform is like choosing a piggy apartment building made of straw. Buildings made of straw require constant upkeep and you are on your own in terms of support.

Running containers on a community distro is like choosing a piggy apartment building made of sticks. It might be slightly more robust / reliable but still comes with no commercial support.



Running containers on a platform like Red Hat Enterprise Linux or OpenShift, Red Hat's container application platform, is like choosing a piggy apartment building made of brick. The platform is supported and maintained by a trusted partner.

Life in the brick apartment complex is best understood through the exploration of the following six characteristics...

1 NAMESPACES

2 RESOURCE CONTROL

3 SECURITY

4 IMAGES

5 OPEN STANDARDS

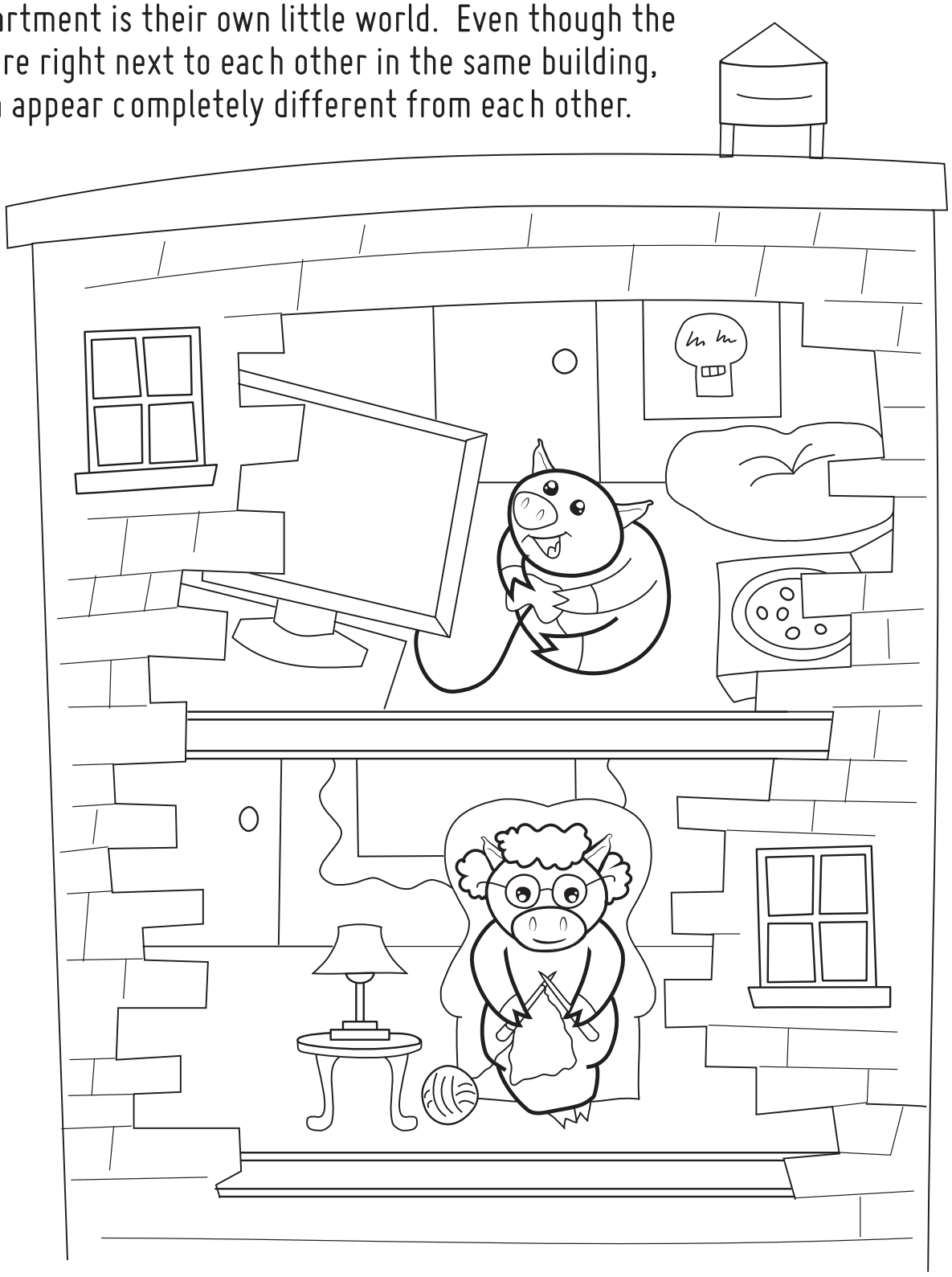
6 MANAGEMENT

NAMESPACES

Our piggy friends who live in apartments share the same building and basic layout. They personalize their space to make it their own.

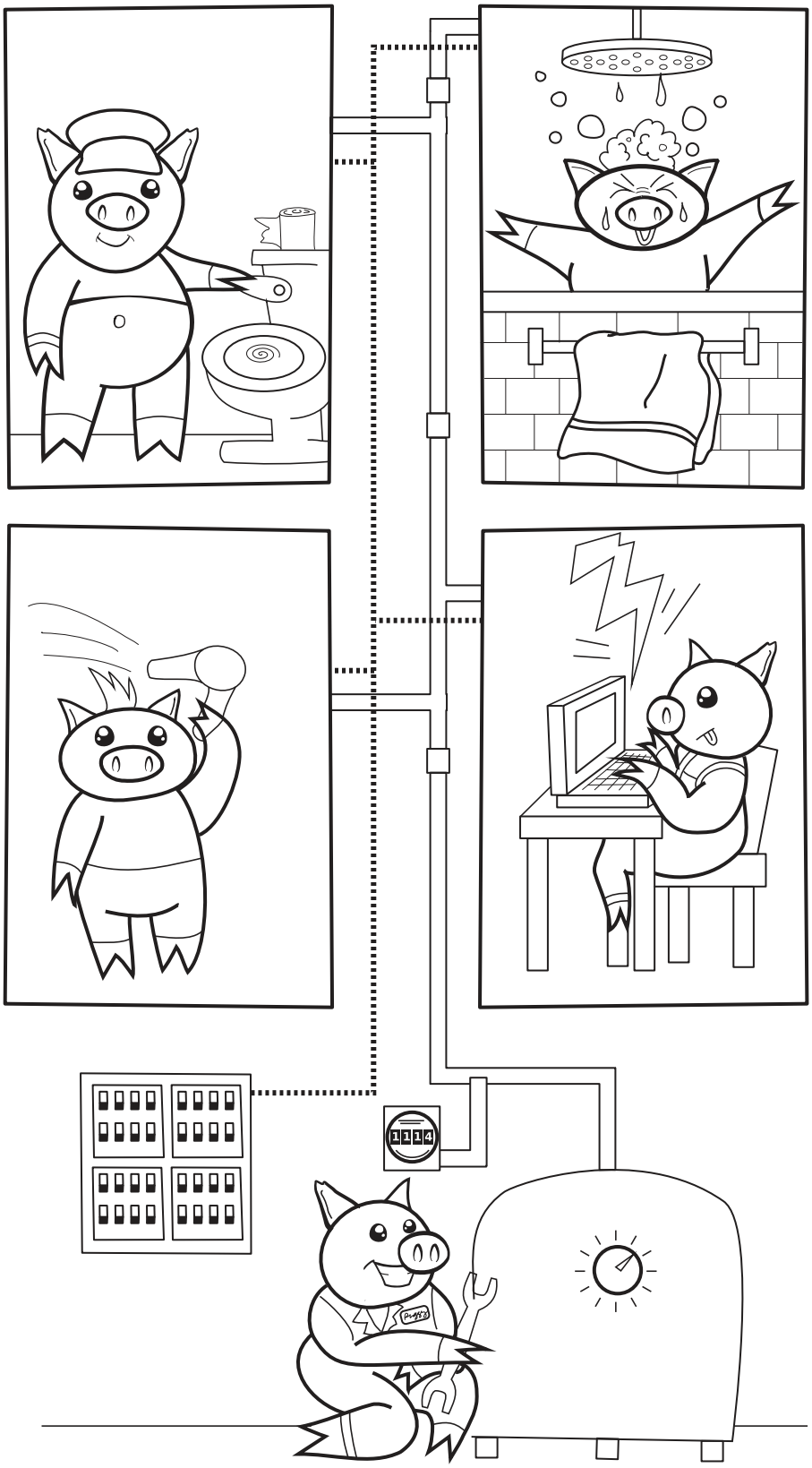
Container namespaces provide containers a way to identify and 'personalize' their own space (as the apartment piggies like to do.)

Each apartment is their own little world. Even though the spaces are right next to each other in the same building, they can appear completely different from each other.



RESOURCE CONTROL

In a shared resource situation, such as piggies sharing an apartment building, resource management is key to a good experience for everyone. For example, flushing the toilet in one apartment should not raise the water temperature in another. Blowing a fuse in one apartment should not kill the power in another.



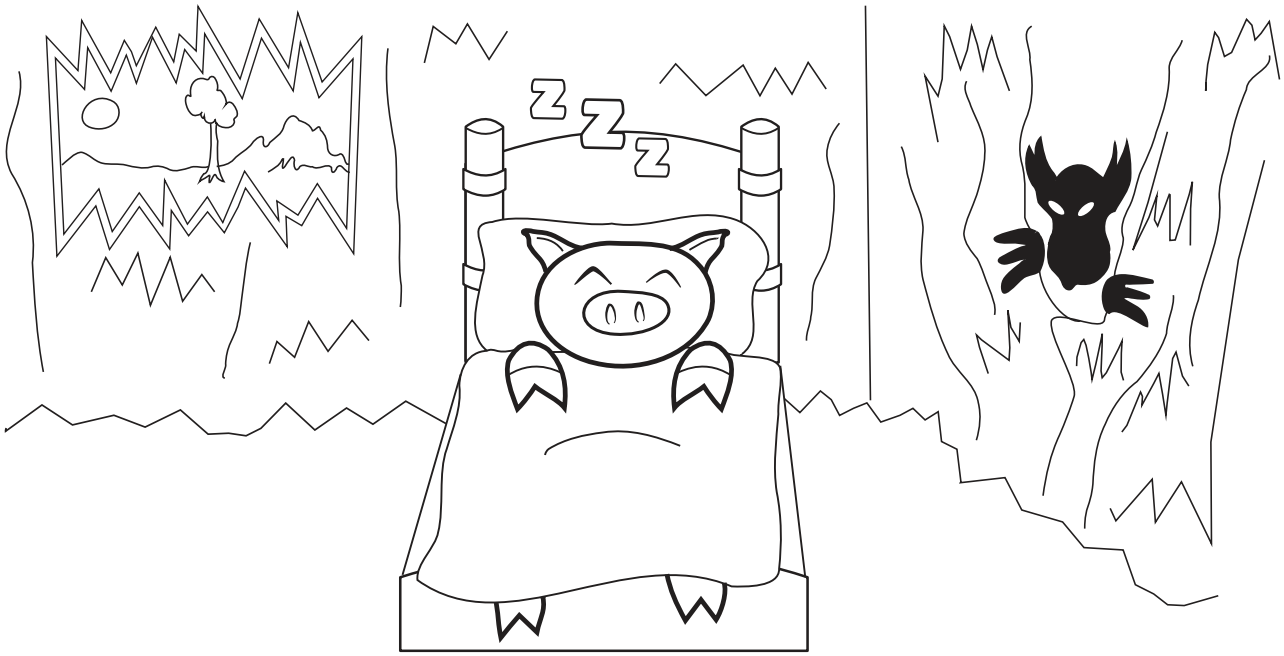
Cgroups are used to manage container resource control. If you have a poorly-written cgroup configuration, you'll run into problems with resources.

In the container world, you want the best performance for shared resources. You can rely on the Red Hat Enterprise Linux kernel for this.

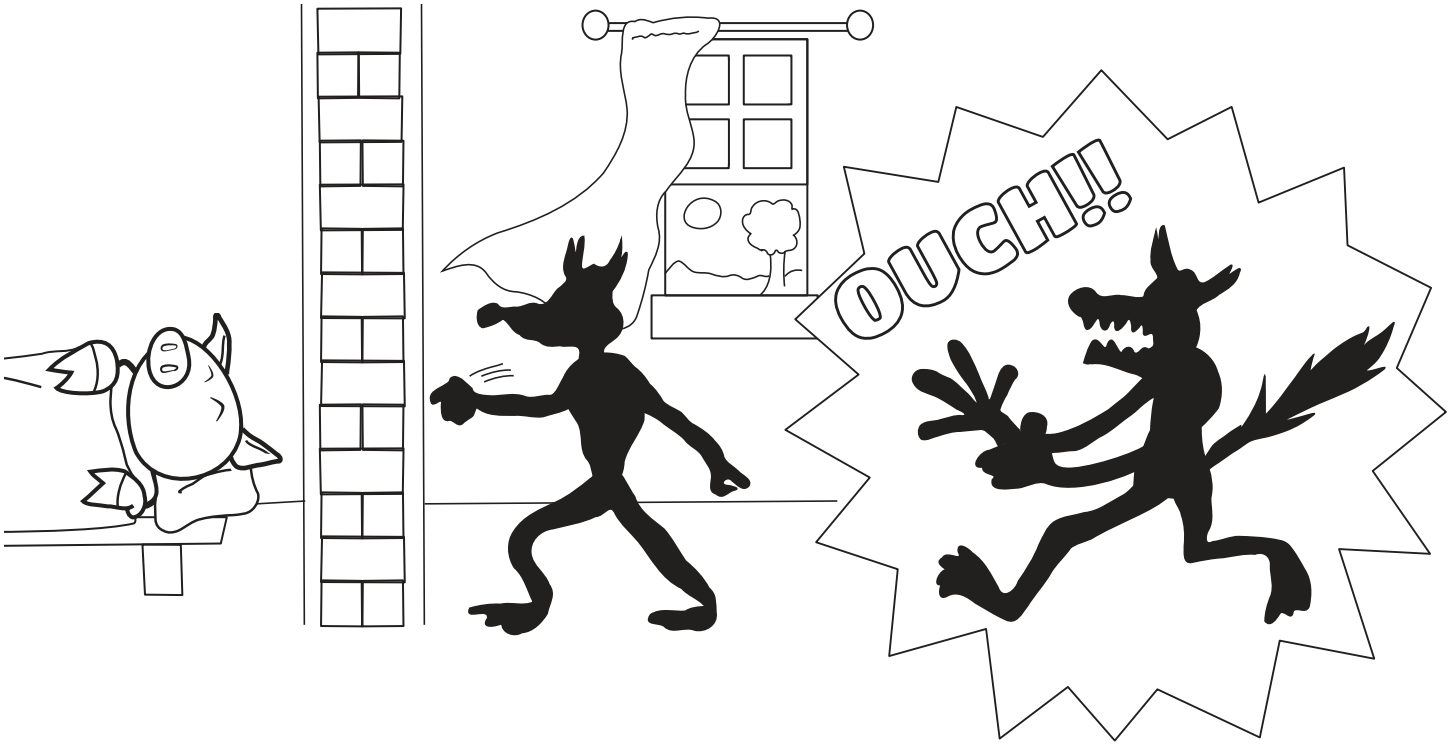
Think of a Red Hat subscription as access to the building super, who makes sure the infrastructure of the building is working correctly and who tunes it as needed.

SECURITY

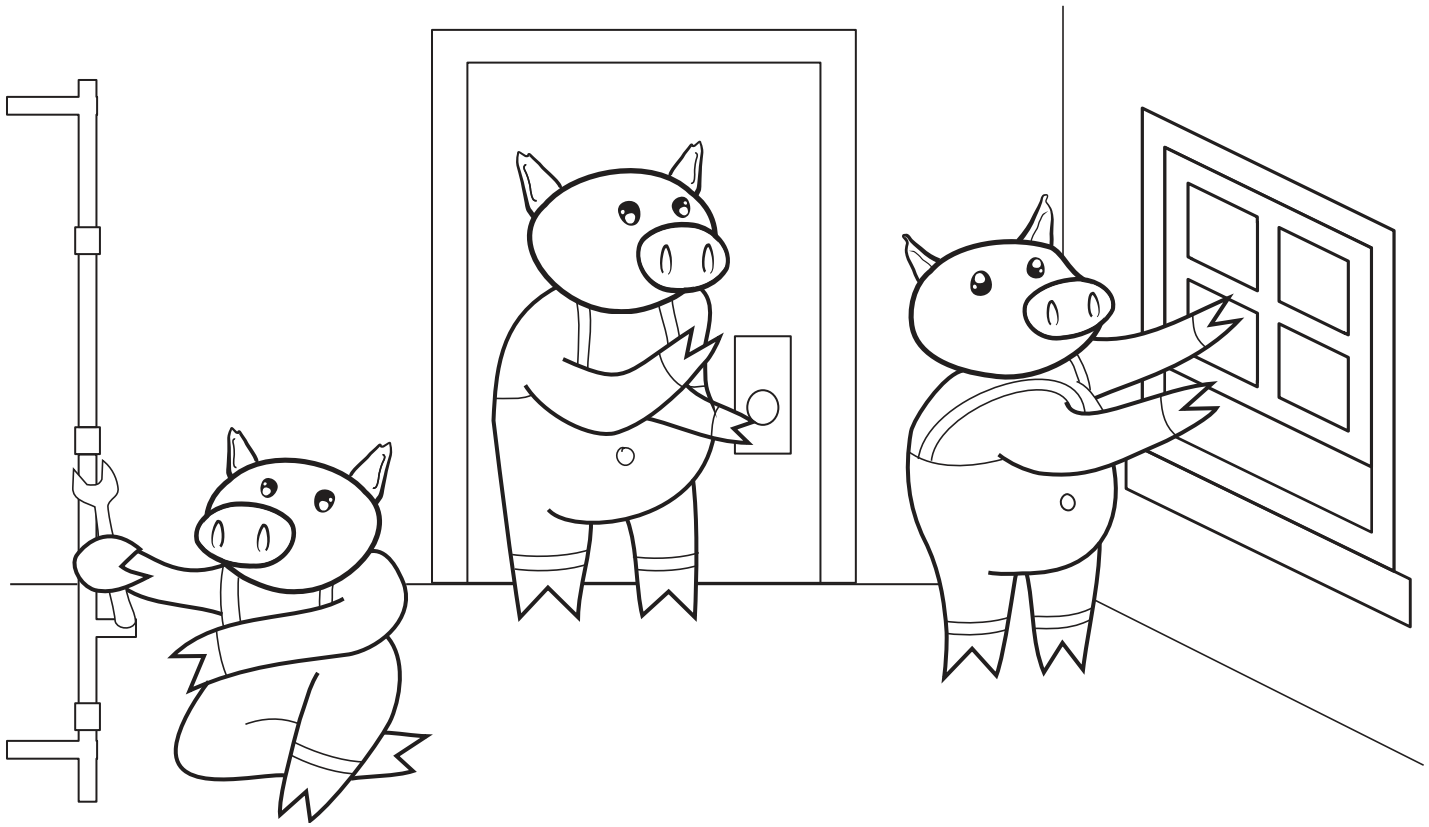
As with apartments, the most secure containers have strong walls between them. You don't want one compromised container to result in the whole system being compromised.



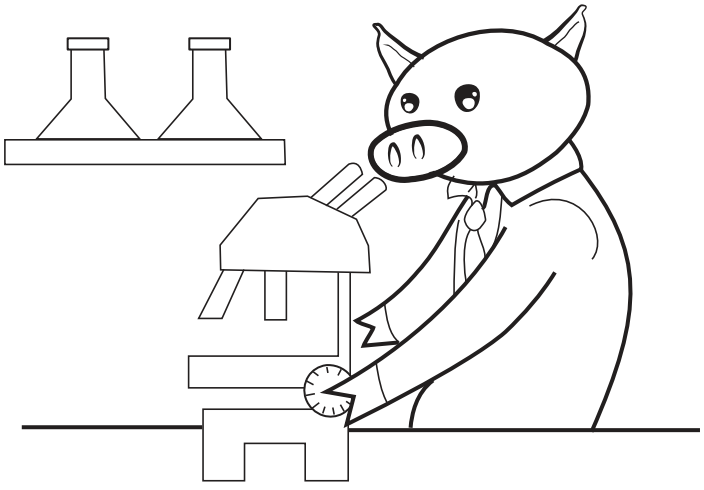
This is very important with containers, because the kernel is shared. What makes the Red Hat "Brick Apartment Building" more secure? SELinux, for one...



Your subscription also gives you access to security analysis tools (like Red Hat's Deep Container Inspection) to scan your containers and hosts for bad configurations and vulnerabilities...



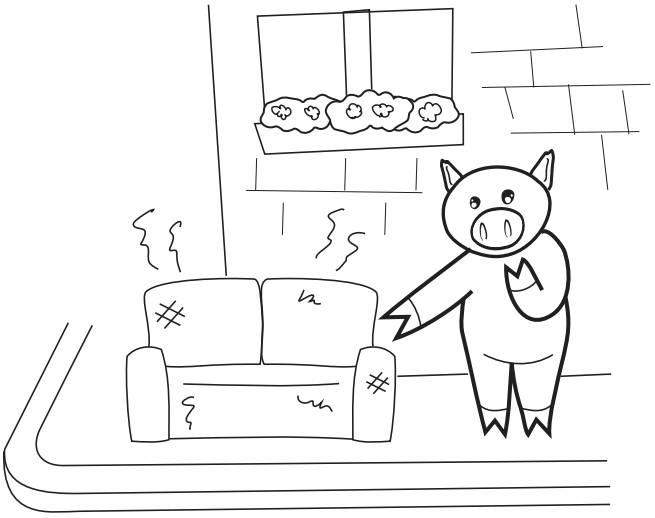
... and access to a team of Red Hat security experts who fix issues as they arise.



Good security practices lower a piggy's risk of an unexpected roast!

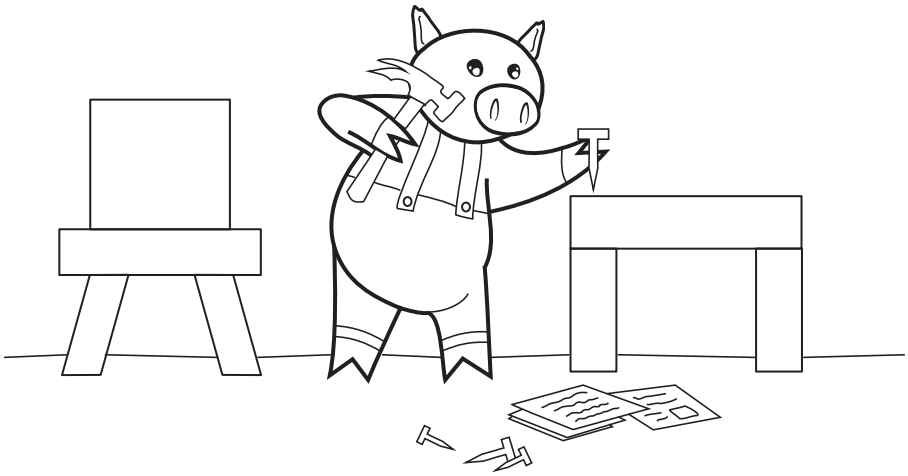
IMAGES

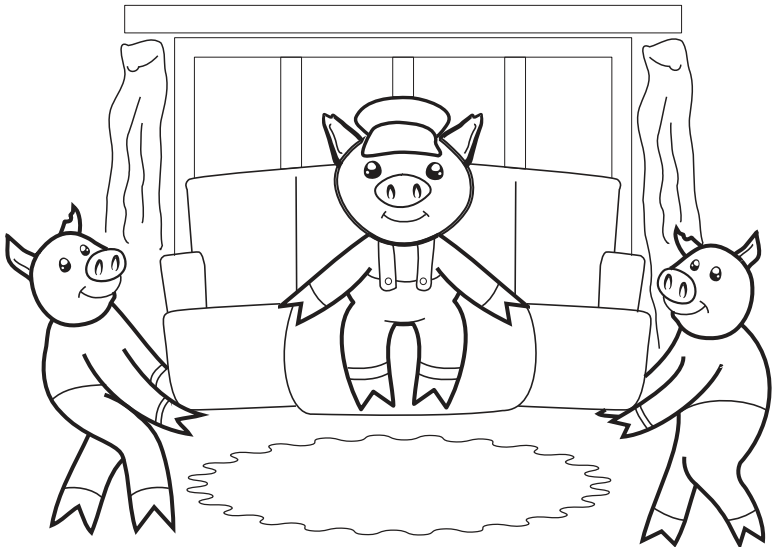
It can be overwhelming to furnish an empty apartment (or container) from scratch.



This piggy sourced some furniture curbside - the safety and cleanliness of such finds is somewhat questionable... almost like picking random container images off the Internet.

This piggy picked up furniture pieces at a warehouse to assemble himself. Pain-staking and time-consuming... almost like building your own base container images.

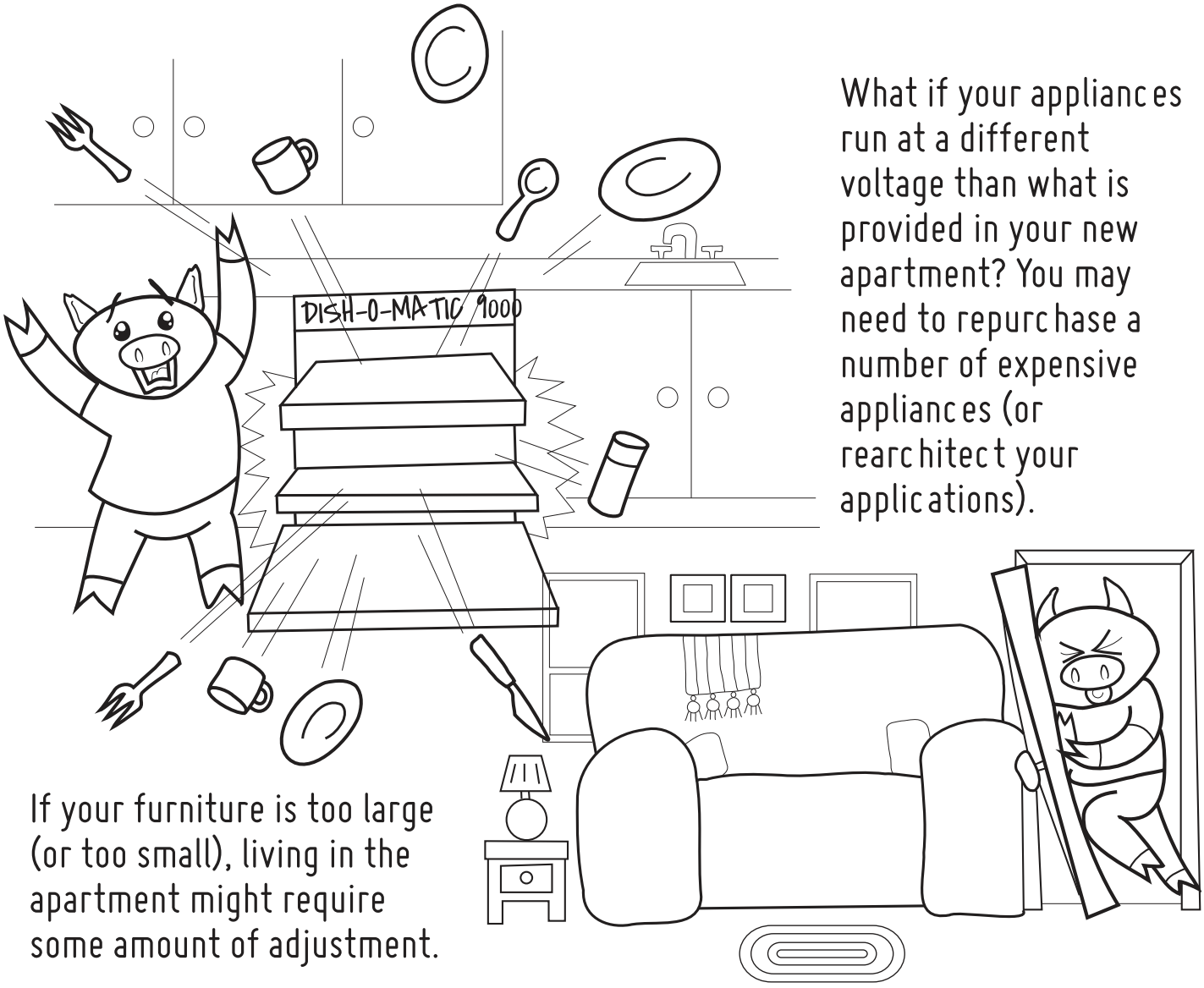




This piggy purchased high-quality, factory-assembled furniture from a showroom and it was delivered to his home via white-glove service. This is like downloading Red Hat certified container images from the Red Hat Registry or from your local Satellite Server.

COMMUNITY STANDARDS

When selecting a piggy apartment building, it's important to ensure that its infrastructure is compliant with common industry standards and policies.



What if your appliances run at a different voltage than what is provided in your new apartment? You may need to repurchase a number of expensive appliances (or rearchitect your applications).

If your furniture is too large (or too small), living in the apartment might require some amount of adjustment.

Standardization and consistency create a common foundation that leads to greater application portability. At Red Hat we always attempt to work with the upstream first. In containers we are the #1 contributor to Docker other than Docker, Inc and #2 in Kubernetes to Google. We also work with the Open Container Initiative and the Cloud Native Computing Foundation to help set and promote shared standards.

Whether it's piggy apartments or Linux containers – infrastructure consistency means you can confidently deploy container-based applications anywhere, from bare metal to cloud environments.

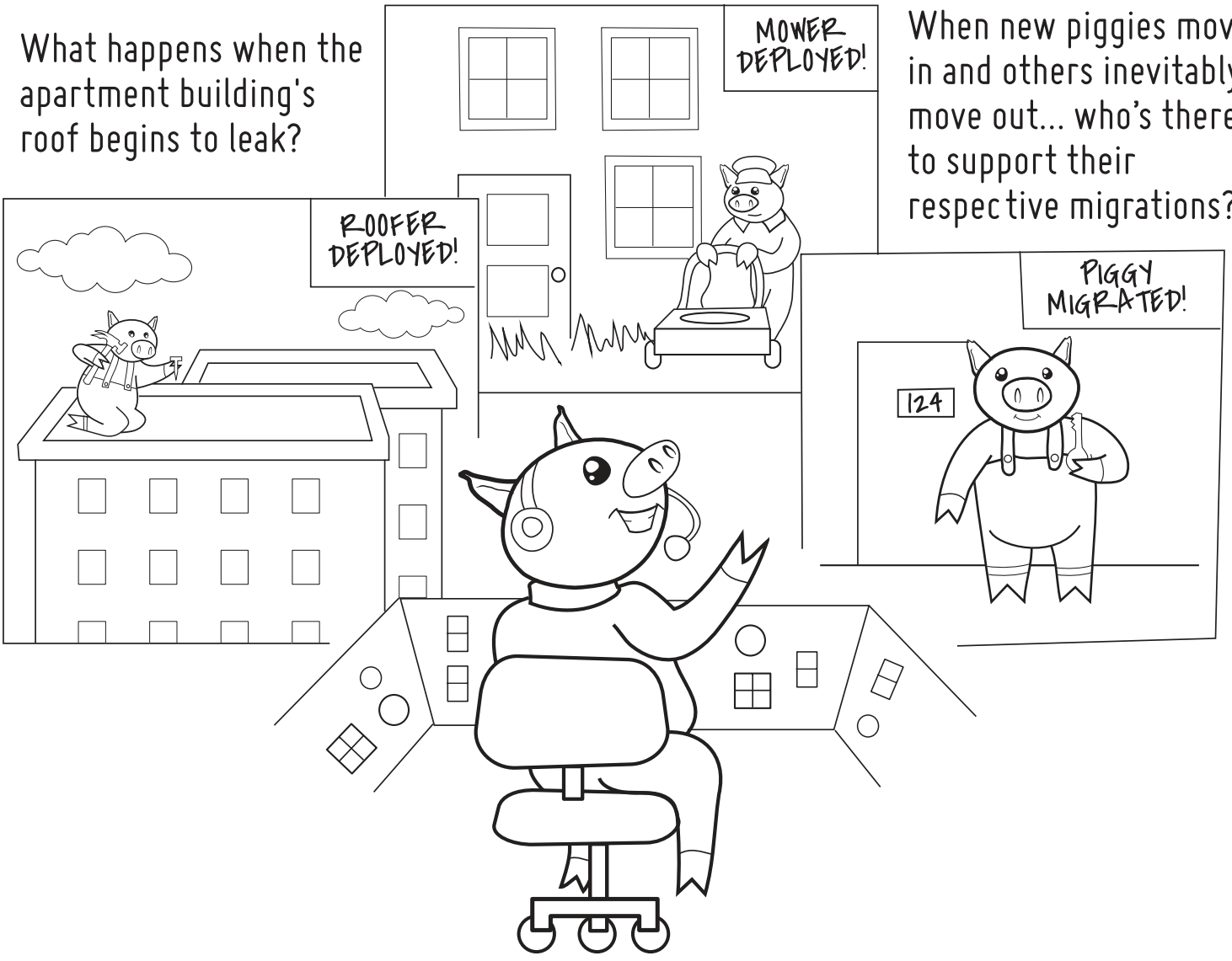
MANAGEMENT

As you expand to house many piggies across many apartment buildings, management and upkeep quickly become complicated and time consuming.

What happens when the lawn becomes overgrown?

What happens when the apartment building's roof begins to leak?

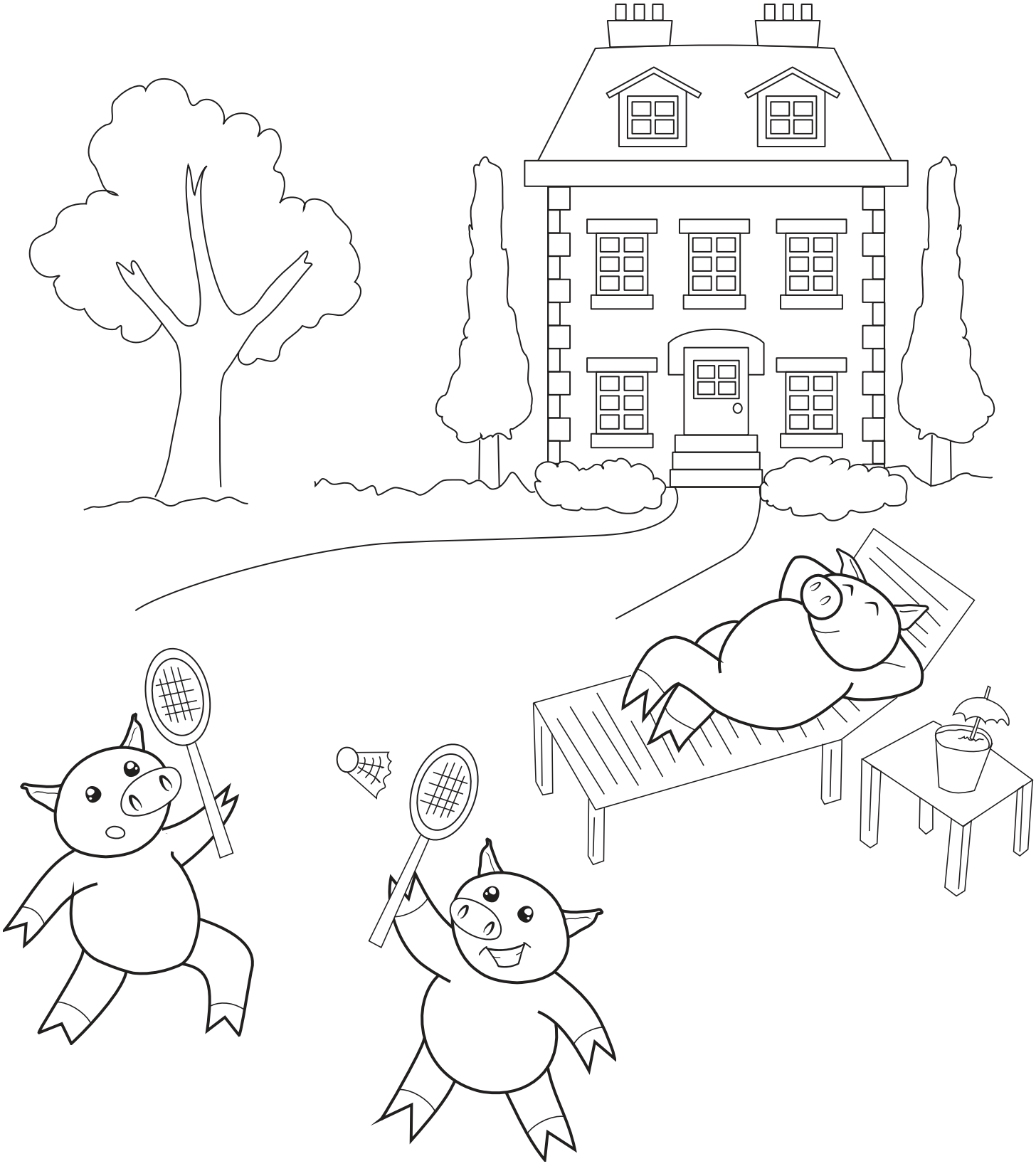
When new piggies move in and others inevitably move out... who's there to support their respective migrations?



Management and upkeep is important with apartments and apartment buildings – especially as you scale up. The same is true for application containers. OpenShift, Red Hat's container platform, works in concert with Red Hat CloudForms to help you streamline node and container creation, deployment, orchestration workflows, and management.

THE END

The piggies have finally found their perfect home. Ready to make the move? Visit **http://red.ht/containers** to learn more.



Learn more at redhat.com:

http://red.ht/containers

Sponsored by Red Hat

LEARN as you COLOR!

the CONTAINER COMMANDOS COLORING BOOK

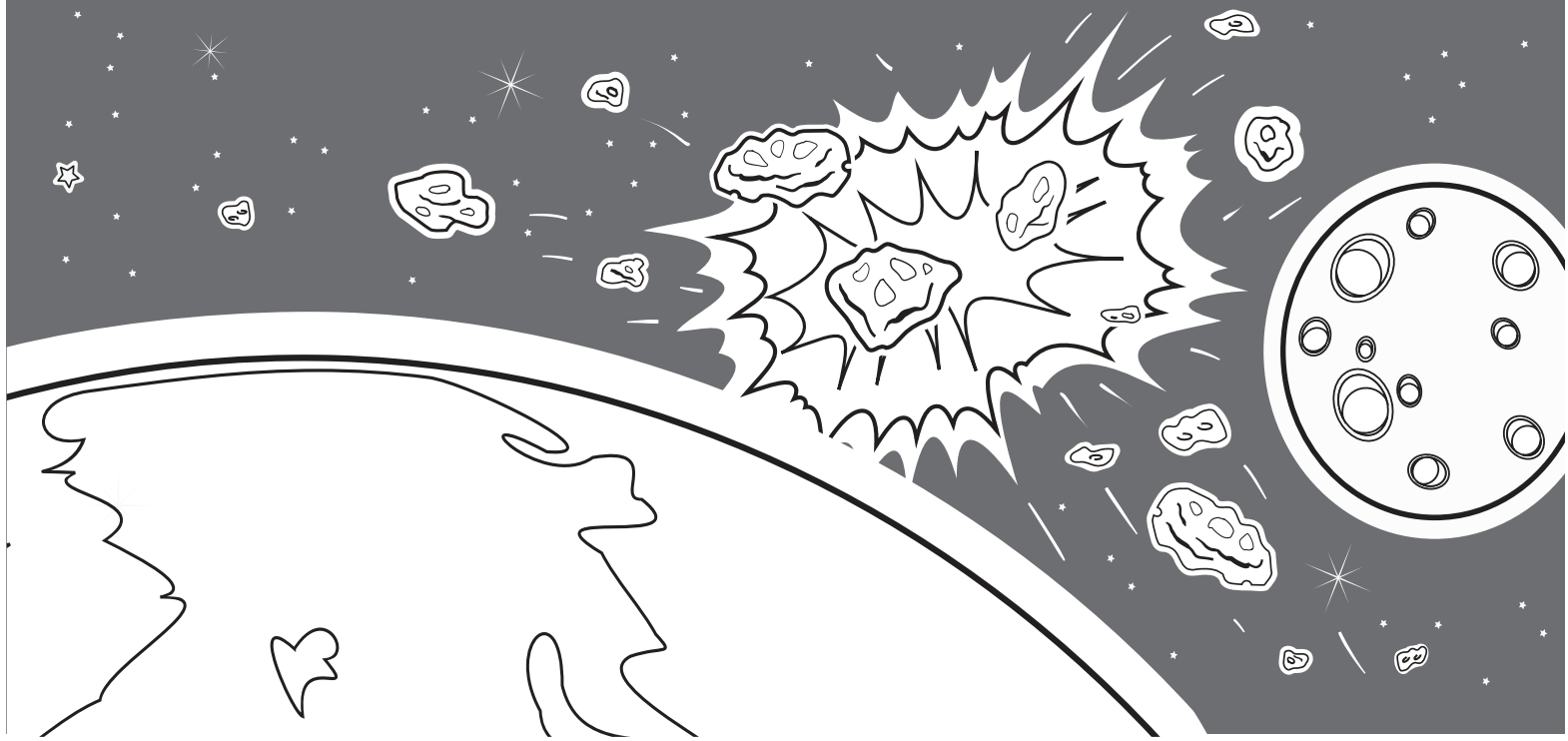
written by
DAN WALSH & MÁIRÍN DUFFY

illustrated by
MÁIRÍN DUFFY

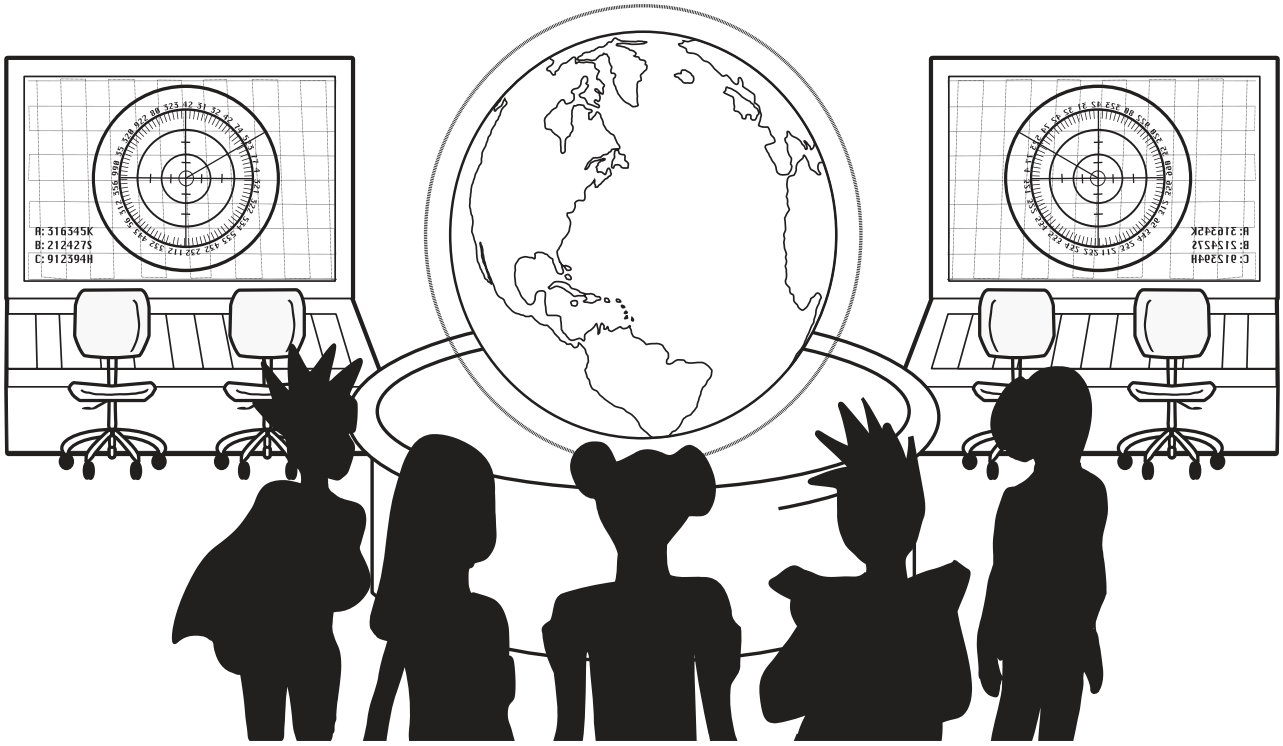
edited by
COLBY HOKE

THE EARTH IS IN DANGER...

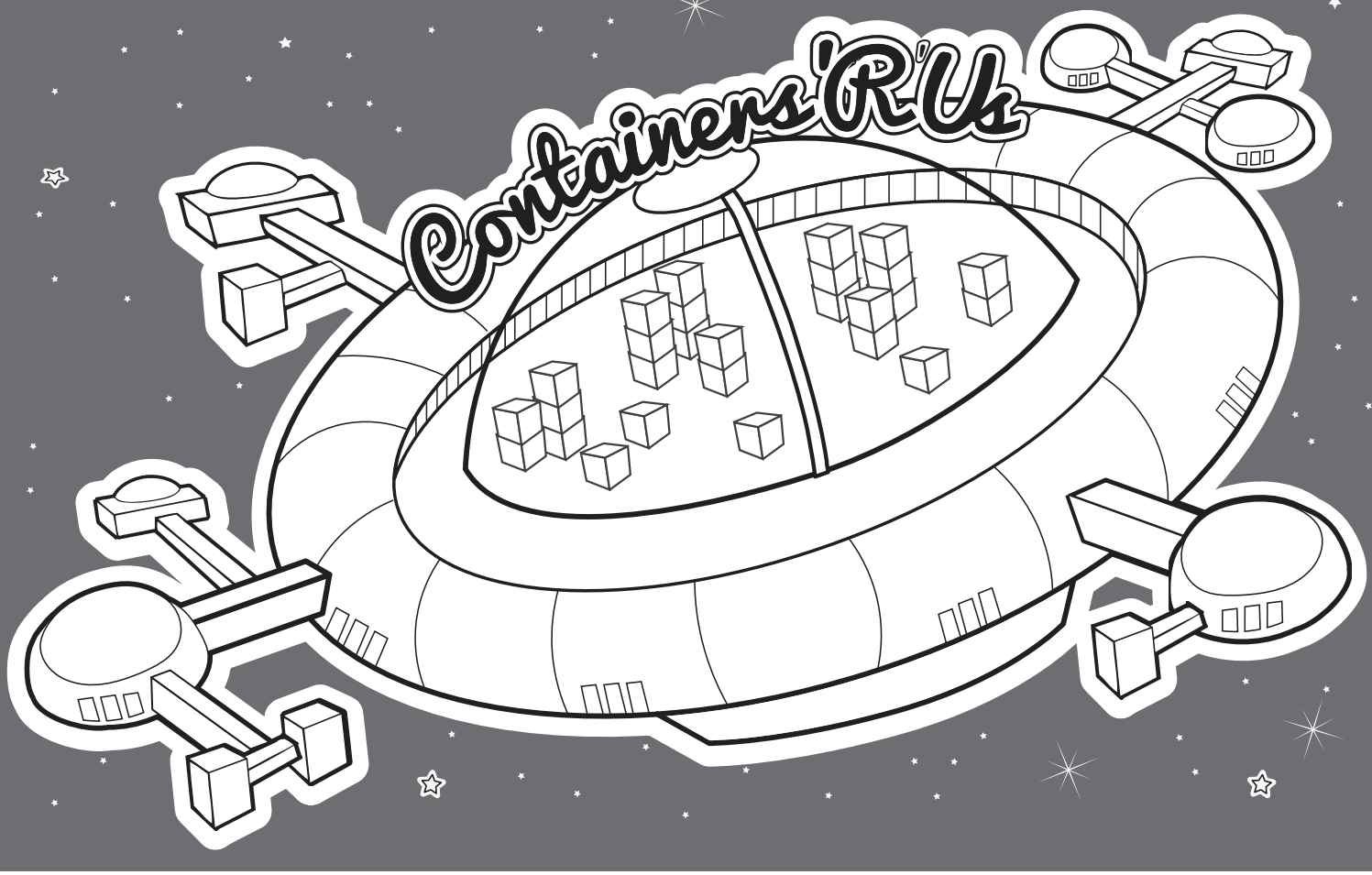
A supercomet exploded in our solar system and rubble is hurtling towards us.

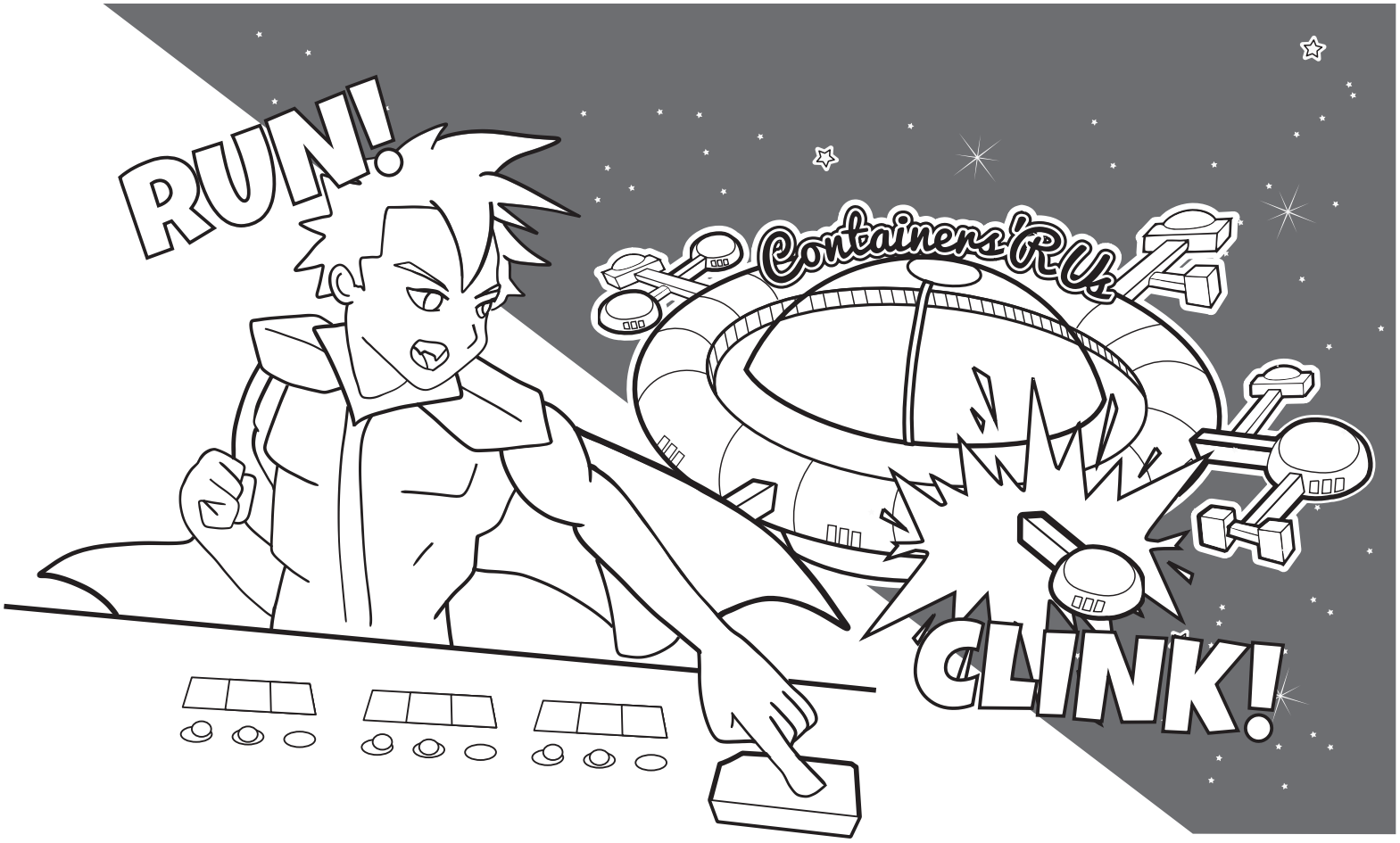


The Global Superhero Alliance must deploy a shield to protect—and save—the planet.

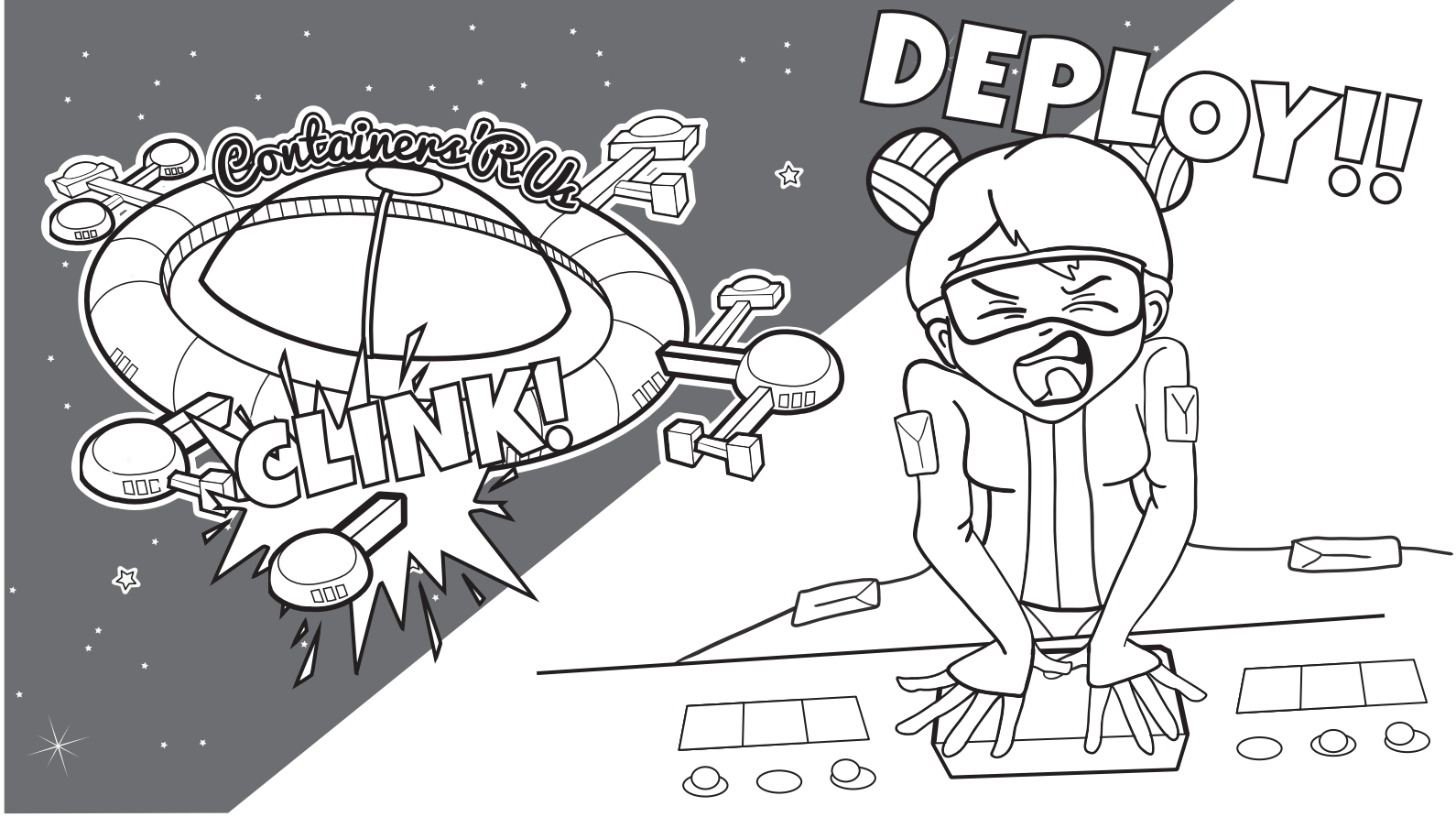


The alliance has partnered with Containers 'R' Us to develop, build, deploy, and manage the container-based, anti-rubble protective shield. Containers 'R' Us delivers containers from a centralized launch station.



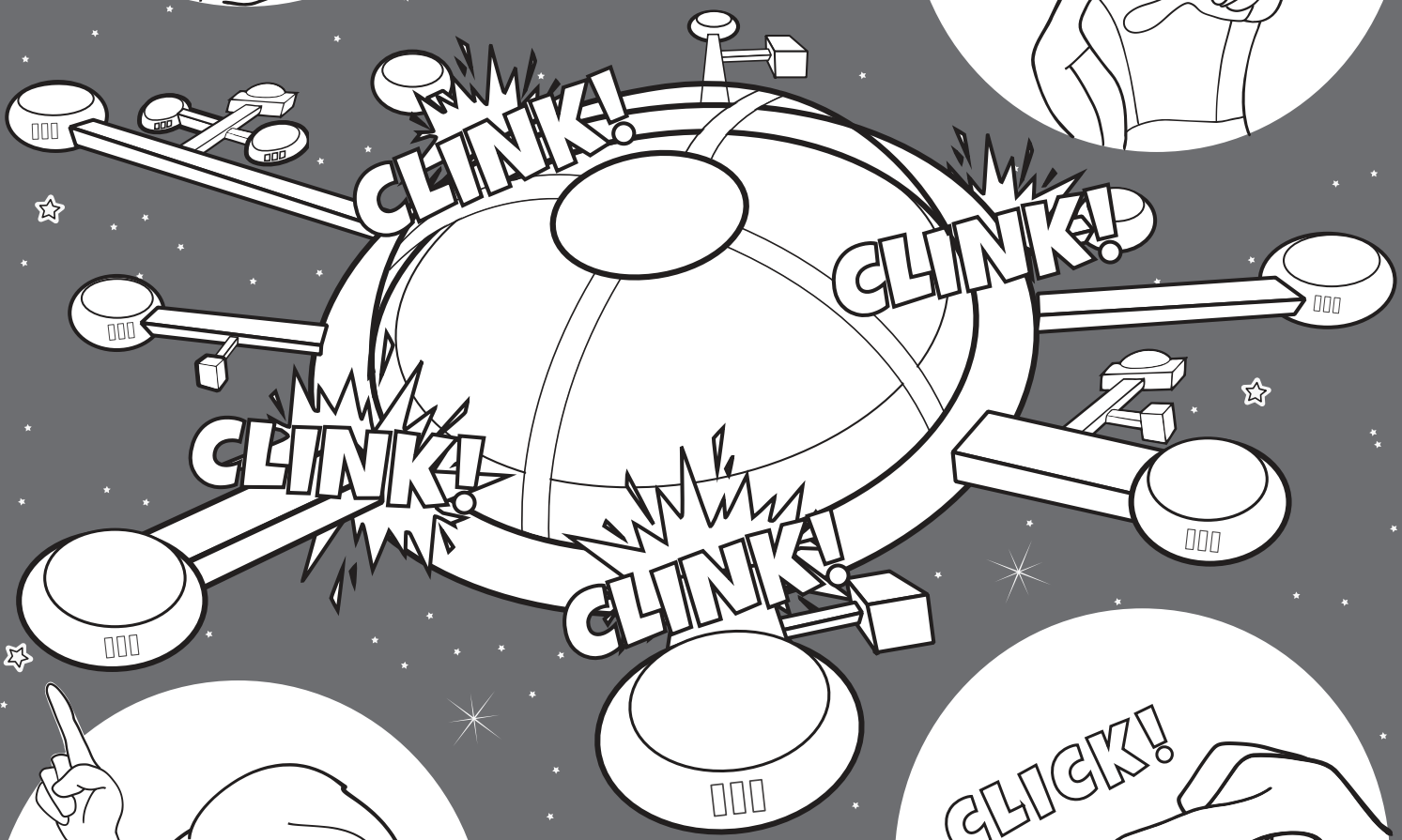


"Containers 'R' Us! Build and deploy the laser-guided targeting container!!"



"LAUNCH THE IMPACT ABSORPTION SHIELD CONTAINER!"

"LAUNCH THE REFLECTIVE SHIELD CONTAINER!"

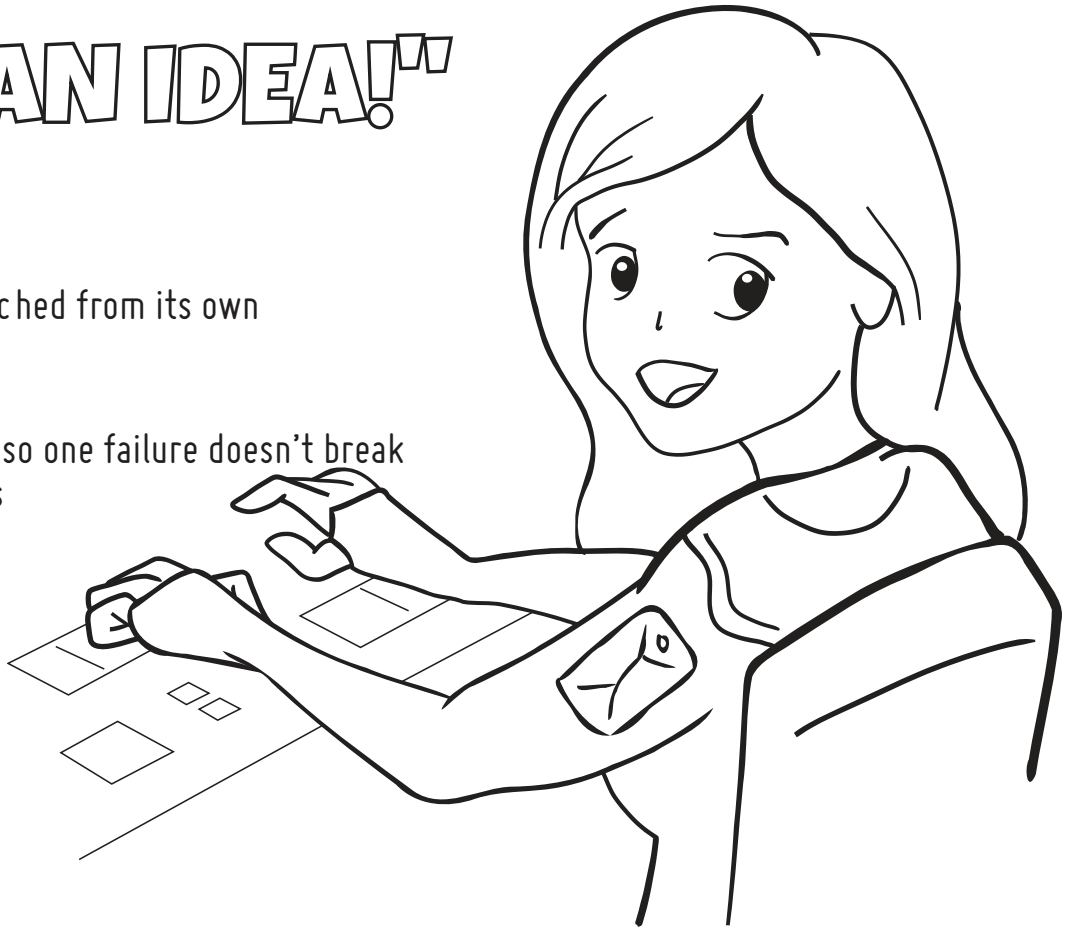


"10 MORE LASER-GUIDED TARGETING CONTAINERS!"

"20 MORE OBJECT DETECTION CONTAINERS!"

WHIRRRRRR...

"I HAVE AN IDEA!"



"What if..."

"...Each container was launched from its own process?"

"....Each tool was separate, so one failure doesn't break all, and we could swap tools based on need?"

"...We could innovate in a broad, distributed, diverse community, instead of relying on a single project arbiter?"

MEET THE

CONTAINER COMMANDOS!

A crack team of individual superheroes, each with strengths tailored for specific container purposes, all collaborating together but no single one overriding all...

PODMAN!

SUPERPOWER:
Low-level, daemonless container image management

A command-line tool for the development of containers. Run standalone, non-orchestrated containers as well as groups of containers called 'pods.' Podman makes it quick, easy, and lightweight to develop, test, and debug containers.

https://github.com/projectatomic/libpod

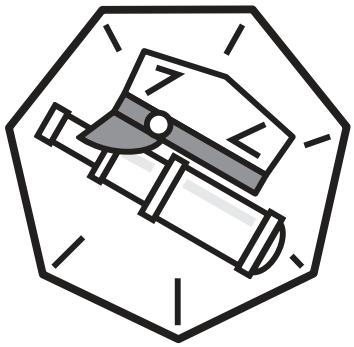
BUILDAH!

SUPERPOWER:
Builds container images

Build both working containers and images in various formats, most notably the open container image (OCI) format. Buildah lets you mount and modify the image's filesystem layer to create a new image. It provides low-level container image constructs so other, higher-level tools can build images in new and interesting ways and it supports Dockerfiles.

https://github.com/projectatomic/buildah

SKOPEO!



SUPERPOWER:
Teleports containers

Skopeo inspects remote container images on registries, but it's more powerful than that. It can copy images between different container image stores or directly into docker daemon or containers/storage—sharing it with CRI-O, buildah, and podman. Skopeo can also convert between Docker and OCI images.

https://github.com/projectatomic/skopeo

CRI-O!

SUPERPOWER:
Runs containers in production

A container runtime based on the Open Container Initiative and the Kubernetes Container Runtime Interface (CRI). Once your containers are ready to run in production, CRI-O can help. It specializes in servicing the needs of the Kubernetes orchestrator—pulling images, creating containers on them, and removing them from the system.

https://github.com/kubernetes-incubator/cri-o

OPENSHIFT!

SUPERPOWER:
Orchestrates containers and container images

OPENSHIFT

Red Hat OpenShift is a container application platform that brings containers to the enterprise. OpenShift includes Kubernetes to automate the deployment, scaling, and management of containerized apps. It also adds developer- and operations-centric tools that enable rapid application development, easy deployment and scaling, and long-term life-cycle maintenance for teams.

https://openshift.io

"We couldn't get a communications line back up to the launch station."

"Thanks to the Container Commandos, we redesigned our container deployment using a decentralized model. Now, one component process going down won't bring everything else with it."

"When the rubble came within range, we were ready—and our system held up!"

The planet is safe!

Separate processes. Specialized projects. Great teamwork.
Check them out for your container projects!

THE END...

Or is it? More adventures await with a new member joining the team—CoreOS! Stay tuned...


SKOPEO
PODMAN
CRI-O
OPENSHIFT
BUILDAH