Qubes Air: Generalizing the Qubes Architecture

21-27 minutes ⋮ 1/21/2018

The Qubes OS project has been around for nearly 8 years now, since its original announcement back in April 2010 (and the actual origin date can be traced back to November 11th, 2009, when an initial email introducing this project was sent within ITL internally). Over these years Qubes has achieved reasonable success: according to our estimates, it has nearly 30k regular users. This could even be considered a great success given that 1) it is a new *operating system*, rather than an *application* that can be installed in the user's favorite OS; 2) it has introduced a (radically?) new approach to managing one's digital life (i.e. an explicit partitioning model into security domains); and last but not least, 3) it has very *specific* hardware requirements, which is the result of using Xen as the hypervisor and Linux-based Virtual Machines (VMs) for networking and USB qubes. (The term "qube" refers to a compartment – not necessarily a VM – inside a Qubes OS system. We'll explain this in more detail below.)

For the past several years, we've been working hard to bring you Qubes 4.0, which features state-of-the-art technology not seen in previous Qubes versions, notably the next generation Qubes Core Stack and our unique Admin API. We believe this new platform (Qubes 4 represents a major rewrite of the previous Qubes codebase!) paves the way to solving many of the obstacles mentioned above.

The new, flexible architecture of Qubes 4 will also open up new possibilities, and we've recently been thinking about how Qubes OS should evolve in the long term. In this article, I discuss this vision, which we call Qubes Air. It should be noted that what I describe in this article has not been implemented yet.

Why?

Before we take a look at the long-term vision, it might be helpful to understand why we would like the Qubes architecture to further evolve. Let us quickly recap some of the most important current weaknesses of Qubes OS (including Qubes 4.0).

Deployment cost (aka "How do I find a Qubes-compatible laptop?")

Probably the biggest current problem with Qubes OS – a problem that prevents its wider adoption – is the difficulty of finding a compatible laptop on which to install it. Then, the whole process of needing to install a new *operating system*, rather than just adding a new *application*, scares many people away. It's hard to be surprised by that.

This problem of deployment is not limited to Qubes OS, by the way. It's just that, in the case of Qubes OS, these problems are significantly more pronounced due to the aggressive use of virtualization technology to isolate not just apps, but also devices, as well as incompatibilities between Linux drivers and modern hardware. (While these driver issues are not inherent to the architecture of Qubes OS, they affected us nonetheless, since we use Linux-based VMs to handle devices.)

The hypervisor as a single point of failure

Since the beginning, we've relied on virtualization technology to isolate individual qubes from one another. However, this has led to the problem of over-dependence on the hypervisor. In recent years, as more and more top notch researchers have begun scrutinizing Xen, a number of security bugs have been discovered. While many of them did not affect the security of Qubes OS, there were still too many that did. :(

Potential Xen bugs present just one, though arguably the most serious, security problem. Other problems arise from the underlying architecture of the x86 platform, where various inter-VM side- and covert-channels are made possible thanks to the aggressively optimized multi-core CPU architecture, most spectacularly demonstrated by the recently published Meltdown and Spectre attacks. Fundamental problems in other areas of the underlying hardware have also been discovered, such as the Row Hammer Attack.

This leads us to a conclusion that, at least for some applications, we would like to be able to achieve better isolation than currently available hypervisors *and* commodity hardware can provide.

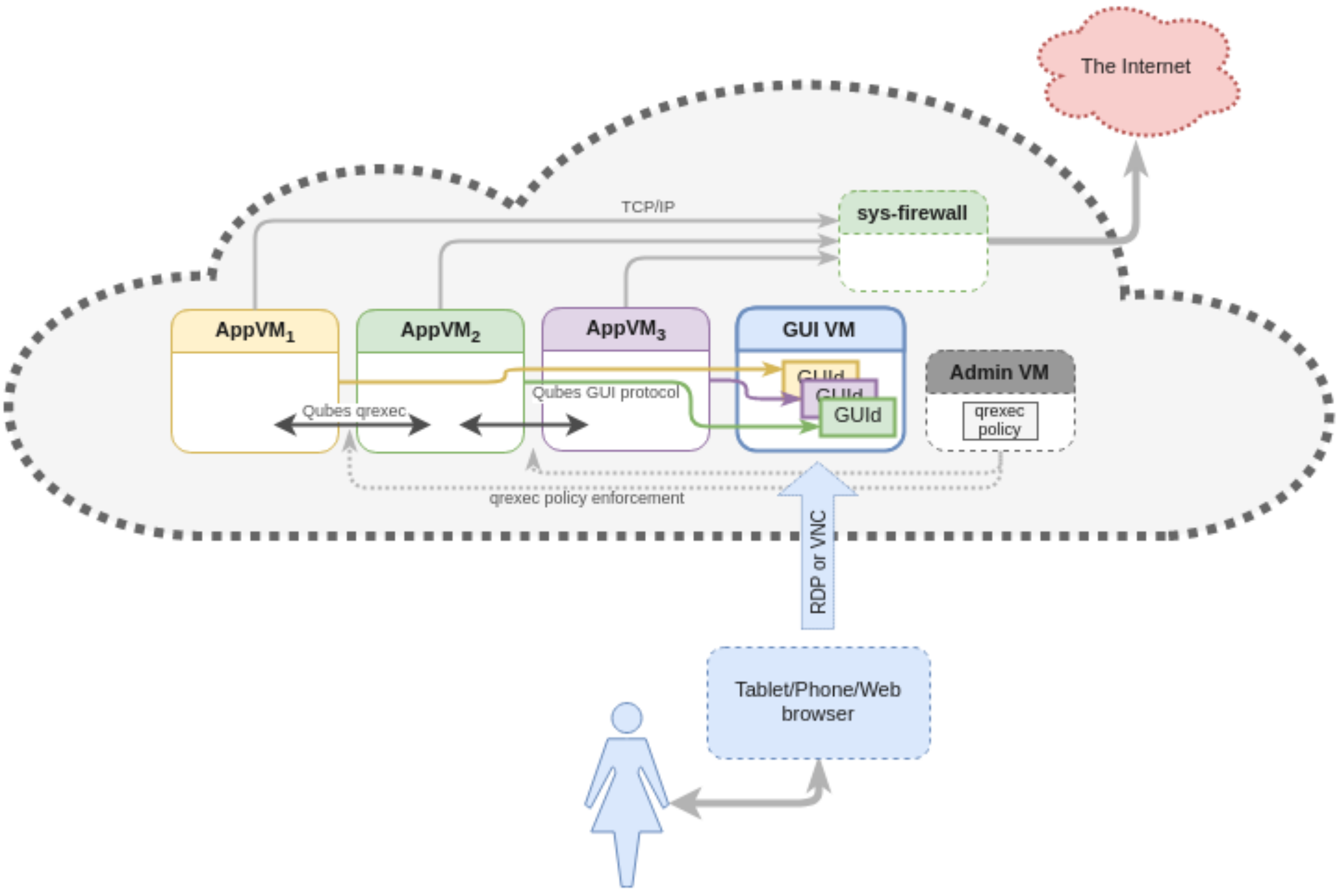
How?

One possible solution to these problems is actually to “move Qubes to the cloud.” Readers who are allergic to the notion of having their private computations running in the (untrusted) cloud should not give up reading just yet. Rest assured that we will also discuss other solutions not involving the cloud. The beauty of Qubes Air, we believe, lies in the fact that all these solutions are largely isomorphic, from both an architecture and code point of view.

Example: Qubes in the cloud

Let’s start with one critical need that many of our customers have expressed: Can we have “Qubes in the Cloud”?

As I’ve emphasized over the years, the essence of Qubes does not rest in the Xen hypervisor, or even in the simple notion of “isolation,” but rather in the careful decomposition of various workflows, devices, apps across securely compartmentalized containers. Right now, these are mostly desktop workflows, and the compartments just happen to be implemented as Xen VMs, but neither of these aspects is essential to the nature of Qubes. Consequently, we can easily imagine Qubes running on top of VMs that are hosted in some cloud, such as Amazon EC2, Microsoft Azure, Google Compute Engine, or even a decentralized computing network, such as Golem. This is illustrated (in a very simplified way) in the diagram below:



It should be clear that such a setup automatically eliminates the deployment problem discussed above, as the user is no longer expected to perform any installation steps herself. Instead, she can access Qubes-as-a-Service with just a Web browser or a mobile app. This approach may trade security for convenience (if the endpoint device used to access Qubes-as-a-Service is insufficiently protected) or privacy for convenience (if the cloud operator is not trusted). For many use cases, however, the ability to access Qubes from any device and any location makes the trade-off well worth it.

We said above that we can imagine “Qubes running on top of VMs” in some cloud, but what exactly does that mean?

First and foremost, we’d want the Qubes Core Stack connected to that cloud’s management API, so that whenever the user executes, say, `qvm-create` (or, more generally, issues any Admin API call, in this case `admin.vm.Create.*`) a new VM gets created and properly connected in the Qubes infrastructure.

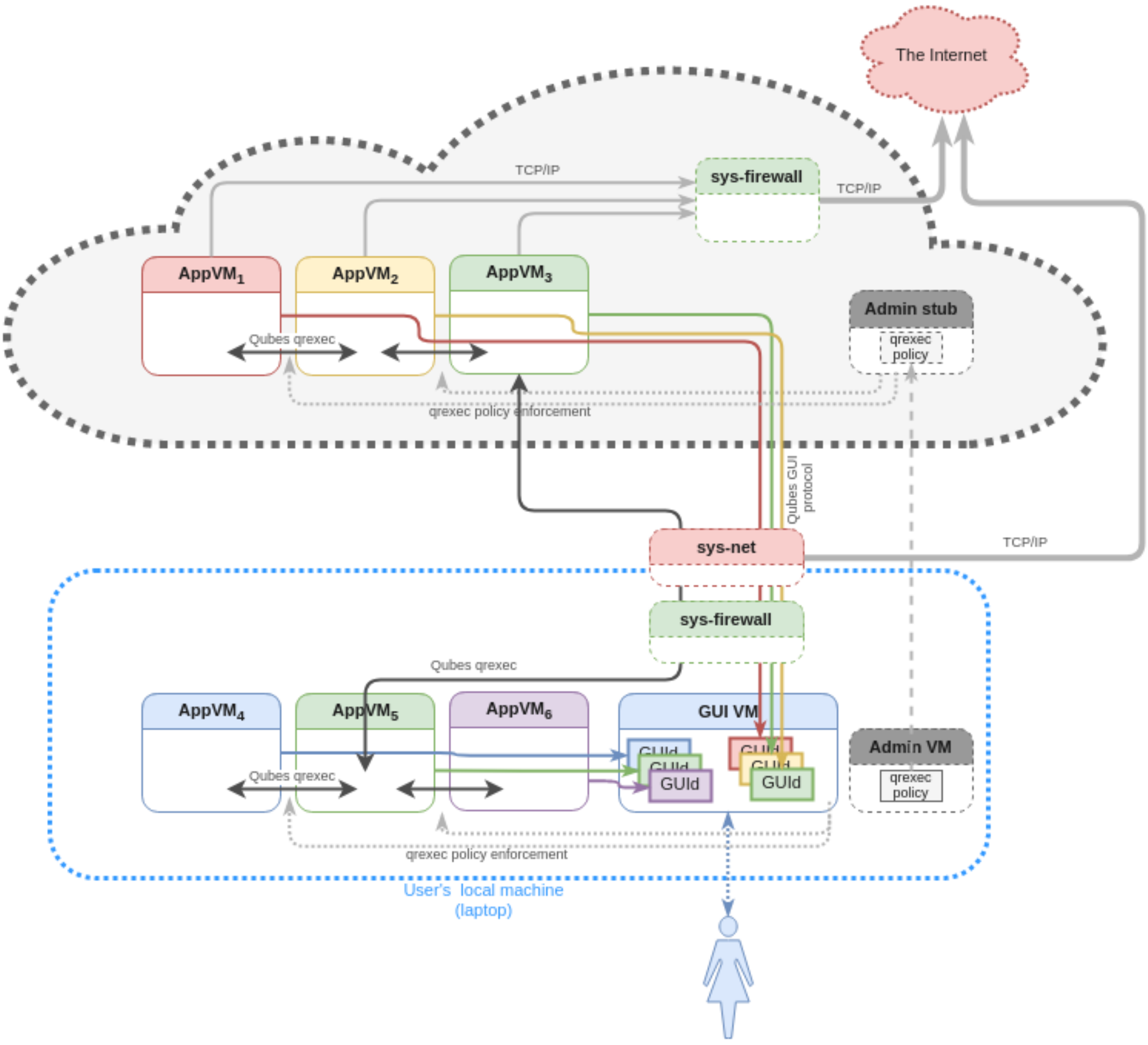
This means that most (all?) Qubes Apps (e.g. Split GPG, PDF and image converters, and many more), which are built around qrexec, should Just Work (TM) when run inside a Qubes-as-a-Service setup.

Now, what about the Admin and GUI domains? Where would they go in a Qubes-as-a-Service scenario? This is an important question, and the answer is much less obvious. We'll return to it below. First, let's look at a couple more examples that demonstrate how Qubes Air could be implemented.

Example: Hybrid Mode

Some users might decide to run a subset of their qubes (perhaps some personal ones) on their local laptops, while using the cloud only for other, less privacy-sensitive VMs. In addition to privacy, another bonus of running some of the VMs locally would be much lower GUI latency (as we discuss below).

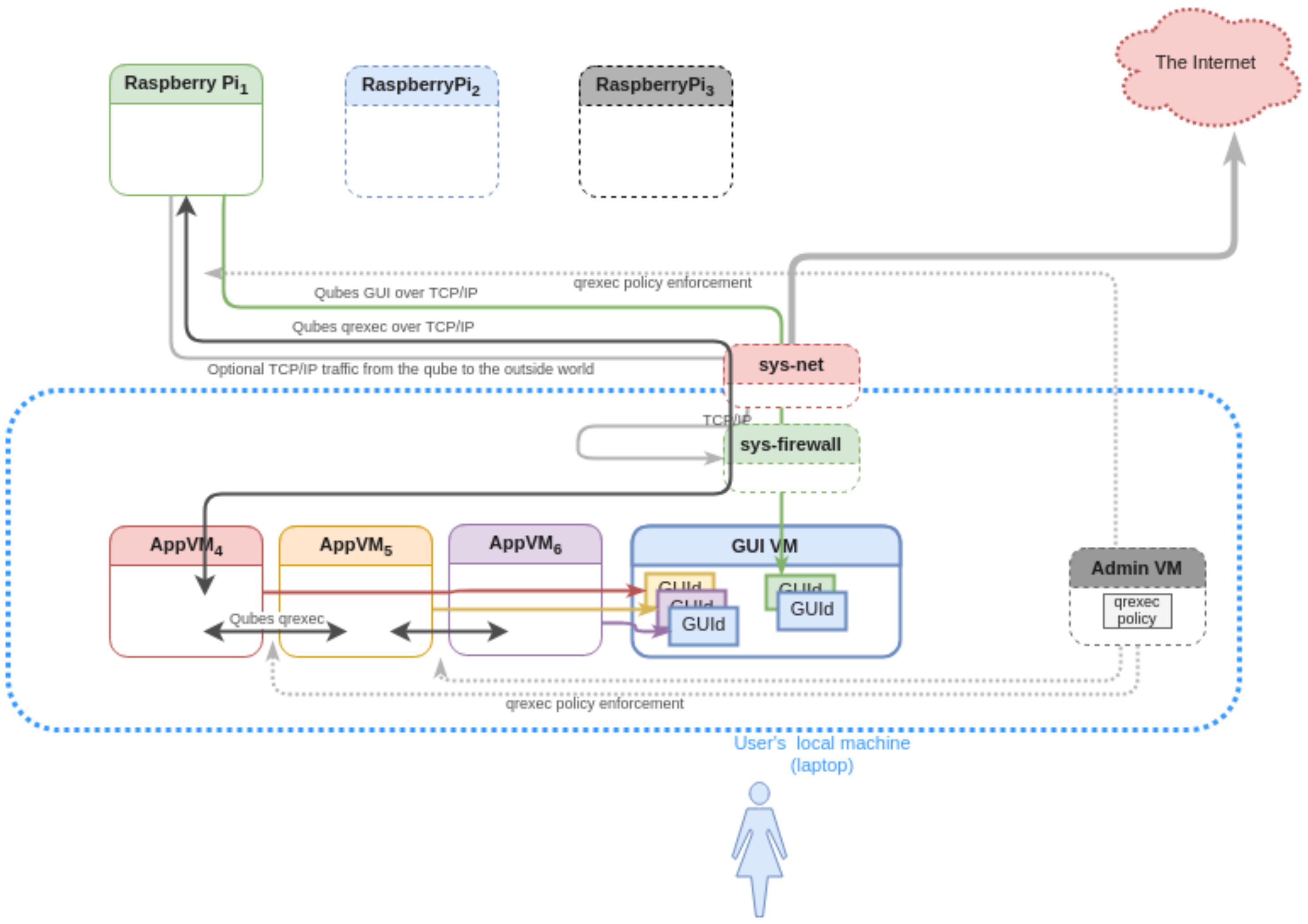
The ability to run some VMs locally and some in the cloud is what I refer to as *Hybrid Mode*. The beauty of Hybrid Mode is that the user doesn't even have to be aware (unless specifically interested!) in whether a particular VM is running locally or in the cloud. The Admin API, qrexec services, and even the GUI, should all automatically handle both cases. Here's an example of a Hybrid Mode configuration:



Another benefit of Hybrid Mode is that it can be used to host VMs across several different cloud providers, not just one. This allows us to solve the problem of over-dependence on a single isolation technology, e.g. on one specific hypervisor. Now, if a fatal security bug is discovered that affects one of the cloud services hosting a group of our VMs, the vulnerability will not automatically affect the security of our other groups of VMs, since the other groups may be hosted on different cloud services, or not in the cloud at all. Crucially, different groups of VMs may be run on different underlying containerization technologies and different hardware, allowing us to diversify our risk exposure against any single class of attack.

Example: Qubes on "air-gapped" devices

This approach even allows us to host each qube (or groups of them) on a physically distinct computer, such as a Raspberry PI or USB Armory. Despite the fact that these are physically separate devices, the Admin API calls, qrexec services, and even GUI virtualization should all work seamlessly across these qubes!



For some users, it may be particularly appealing to host one's Split GPG backend or password manager on a physically separate qube. Of course, it should also be possible to run normal GUI-based apps, such as office suites, if one wants to dedicate a physically separate qube to work on a sensitive project.

The ability to host qubes on distinct physical devices of radically different kinds opens up numerous possibilities for working around the security problems with hypervisors and processors we face today.

Under the hood: Qubes Zones

We've been thinking about what changes to the current Qubes architecture, especially to the Qubes Core Stack, would be necessary to make the scenarios outlined above easy (and elegant) to implement.

There is one important new concept that should make it possible to support all these scenarios with a unified architecture. We’ve named it **Qubes Zones**.

A **Zone** is a concept that combines several things together:

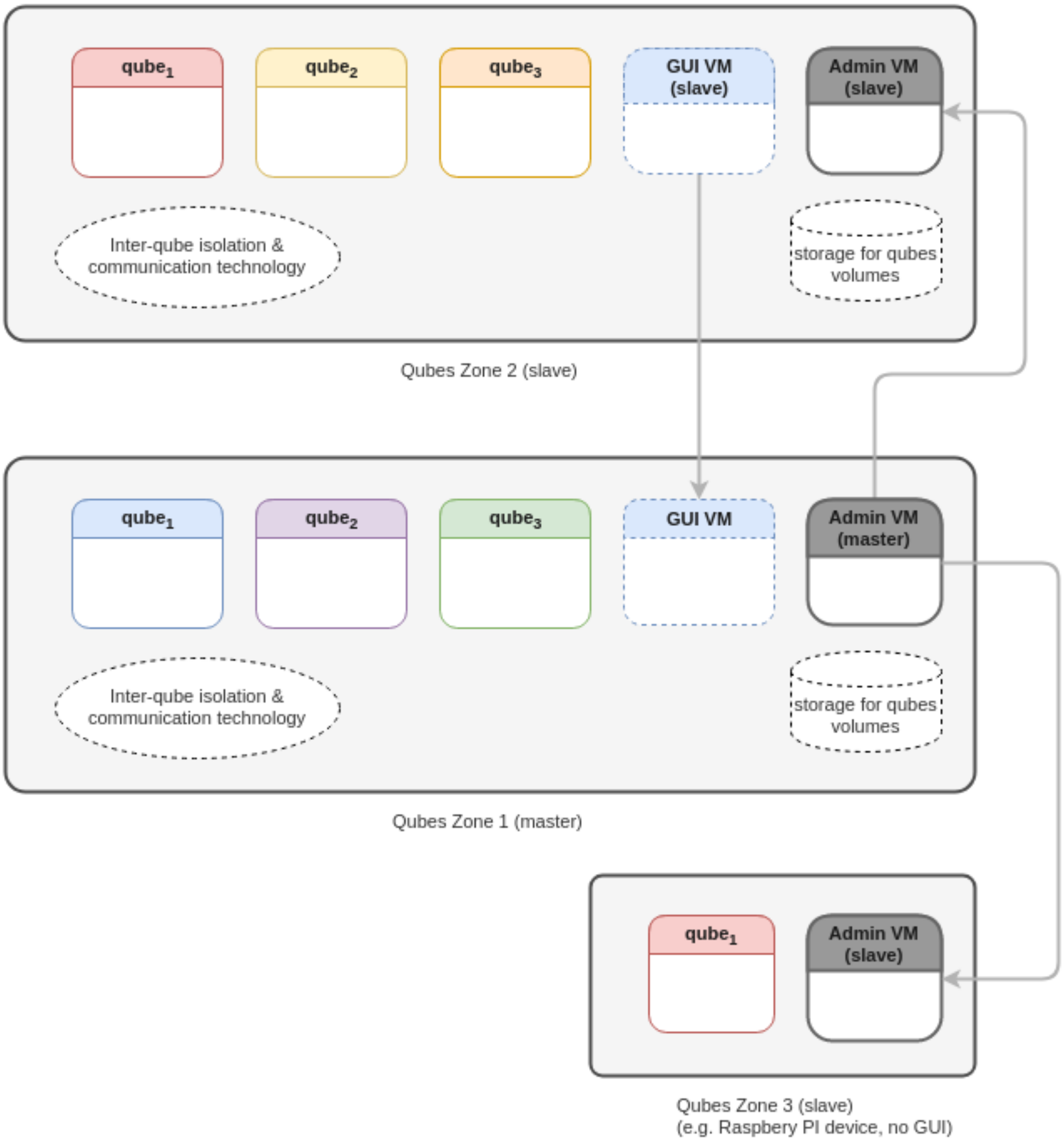
- An underlying “isolation technology” used to implement qubes, which may or may not be VMs. For example, they could be Raspberry PIs, USB Armory devices, Amazon EC2 VMs, or Docker containers.

- The inter-qube communication technology. In the case of qubes implemented as Xen-based VMs (as in existing Qubes OS releases), the Xen-specific shared memory mechanism (so called Grant Tables) is used to implement the communication between qubes. In the case of Raspberry PIs, Ethernet technology would likely be used. In the case of Qubes running in the cloud, some form of cloud-provided networking would provide inter-qube communication. Technically speaking, this is about how Qubes’ vchan would be implemented, as the qrexec layer should remain the same across all possible platforms.

- A “local copy” of an *Admin qube* (previously referred to as the “AdminVM”), used mainly to orchestrate VMs and make policing decisions for all the qubes within the Zone. This Admin qube can be in either “Master” or “Slave” mode, and there can only be one Admin qube running as Master across all the Zones in *one Qubes system*.

- Optionally, a “local copy” of *GUI qube* (previously referred to as the “GUI domain” or “GUIVM”). As with the Admin qube, the GUI qube runs in either Master or Slave mode. The user is expected to connect (e.g. with the RDP protocol) or log into the GUI qube that runs in Master mode (and only that one), which has the job of combining all the GUI elements exposed via the other GUI qubes (all of which must run in Slave mode).

- Some technology to implement storage for the qubes running within the Zone. In the case of Qubes OS running Xen, the local disk is used to store VM images (more specifically, in Qubes 4.0 we use Storage Pools by default). In the case of a Zone composed of a cluster of Raspberry PIs or similar devices, the storage could be a bunch of micro-SD cards (each plugged into one Raspberry PI) or some kind of network storage.



So far, this is nothing radically new compared to what we already have in Qubes OS, especially since we have nearly completed our effort to abstract the Qubes architecture away from Xen-specific details – an effort we code-named *Qubes Odyssey*.

What *is* radically different is that we now want to allow more than one Zone to exist in a single Qubes system!

In order to support multiple Zones, we have to provide transparent proxying of qrexec services across Zones, so that a qube need not be aware that another qube from which it requests a service resides in a different zone. This is the main reason we've introduce multiple "local" Admin qubes – one for each Zone. Slave Admin qubes are also bridges that allow the Master Admin qube to manage the whole system (e.g. request the creation of new qubes, connect and set up storage for qubes, and set up networking between qubes).

Under the hood: qubes' interfaces

Within one Zone, there are multiple *qubes*. Let me stress that the term "qube" is very generic and does not imply any specific technology. It could be a VM under *some* virtualization system. It could be some kind of a container or a physically separate computing device, such as a Raspberry PI, Arduino board, or similar device.

While a qube can be implemented in many different ways, there are certain features it should have:

1. A qube should implement a vchan endpoint. The actual technology on top of which this will be implemented – whether some shared memory within a virtualization or containerization system, TCP/IP, or something else – will be specific to the kind of Zone it occupies.

2. A qube should implement a qrexec endpoint, though this should be very straightforward if a vchan endpoint has already been implemented. This ensures that most (all?) the qrexec services, which are the basis for most of the integration, apps, and services we have created for Qubes, should Just Work(TM).

3. Optionally, for some qubes, a GUI endpoint should also be implemented (see the discussion below).

4. In order to be compatible with Qubes networking, a qube should expect *one* uplink network interface (to be exposed by the management technology specific to that particular Zone), and (optionally) multiple downlink network interfaces (if it is to work as a proxy qube, e.g. VPN or firewalling qube).

5. Finally, a qube should expect two kinds of volumes to be exposed by the Zone-specific management stack:

    - one read-only, which is intended to be used as a root filesystem by the qube (the management stack might also expose an auxiliary volume for implementing copy-on-write illusion for the VM, like the `volatile.img` we currently expose on Qubes),
    - and one read-writable, which is specific to this qube, and which is intended to be used as home directory-like storage. This is, naturally, to allow the implementation of Qubes templates, a mechanism that we believe brings not only a lot of convenience but also some security benefits.

GUI virtualization considerations

Since the very beginning, Qubes was envisioned as a system for desktop computing (as opposed to servers). This implied that GUI virtualization was part of the core Qubes infrastructure.

However, with some of the *security-optimized* management infrastructure we have recently added to Qubes OS, i.e. Salt stack integration (which significantly shrinks the attack surface on the system TCB compared to more traditional "management" solutions), the Qubes Admin API (which allows for the fine-grained decomposition of management roles), and deeply integrated features such as templates, we think Qubes Air may also be useful in some non-desktop applications, such as the embedded appliance space, and possibly even on the server/services side. In this case, it makes perfect sense to have qubes not implement GUI protocol endpoints.

However, I still think that the primary area where Qubes excels is in securing desktop workflows. For these, we need GUI ~~virtualization~~multiplexing, and the qubes need to implement GUI protocol endpoints. Below, we discuss some of the trade-offs involved here.

The Qubes GUI protocol is optimized for security. This means that the protocol is designed to be extremely simple, allowing only for very simple processing on incoming packets, thus significantly limiting the attack surface on the GUI daemon (which is usually considered trusted). The price we pay for this security is the lack of various optimizations, such as on-the-fly compression, which others protocols, such as VNC and RDP, naturally offer. So far, we've been able to get away with these trade-offs, because in current Qubes releases the GUI protocol runs over Xen shared memory. DRAM is very fast (i.e has low latency and super-high bandwidth), and the implementation on Xen smartly makes use of page *sharing* rather than memory *copying*, so that it achieves near native speed (of course with the limitation that we don't expose GPU functionalities to VMs, which might limit the experience in some graphical applications anyway).

However, when qubes run on remote computers (e.g in the cloud) or on physically separate computers (e.g. on a cluster of Raspberry PIs), we face the potential problem of graphics performance. The solution we see is to introduce a local copy of the GUI qube into each zone. Here, we make the assumption that there should be a significantly faster communication channel available between qubes within a Zone than between Zones. For example, inter-VM communication within one data center should be significantly faster than between the user's laptop and the cloud. The Qubes GUI protocol is then used between qubes and the local GUI qube within a single zone, but a more efficient (and more complex) protocol is used to aggregate the GUI into the Master GUI qube from all the Slave GUI qubes. Thanks to this combined setup, we still get the benefit of a reasonably secure GUI. Untrusted qubes still use the Qubes secure GUI protocol to communicate with the local GUI qube. However, we also benefit from the greater efficiency of remote access-optimized protocols such as RDP and VNC to get the GUI onto the user's device over the network. (Here, we make the assumption that the Slave GUI qubes are significantly more trustworthy than other non-privileged qubes in the Zone. If that's not the case, *and* if we're also worried about an attacker who has compromised a Slave GUI qube to exploit a potential bug in the VNC or RDP protocol in order to attack the Master GUI qube, we could still resort to the fine-grained Qubes Admin API to limit the potential damage the attacker might inflict.)

Digression on the "cloudification" of apps

It's hard not to notice how the model of desktop applications has changed over the past decade or so, where many standalone applications that previously ran on desktop computers now run in the cloud and have only their frontends executed in a browser running on the client system. How does the Qubes compartmentalization model, and more importantly Qubes as a *desktop* OS, deal with this change?

Above, we discussed how it's possible to move Qubes VMs from the user's local machine to the cloud (or to physically separate computers) without the user having to notice. I think it will be a great milestone when we finally get there, as it will open up many new applications, as well as remove many obstacles that today prevent the easy deployment of Qubes OS (such as the need to find and maintain dedicated hardware).

However, it's important to ask ourselves how relevant this model will be in the coming years. Even with our new approach, we're still talking about classic standalone desktop applications running in qubes, while the rest of the world seems to be moving toward an app-as-a-service model in which everything is hosted in the cloud (e.g. Google Docs and Microsoft Office 365). How relevant is the whole Qubes architecture, even the cloud-based version, in the app-as-a-service model?

I'd like to argue that the Qubes architecture still makes perfect sense in this new model.

First, it's probably easy to accept that there will always be applications that users, both individual and corporate, will prefer (or be forced) to run locally, or at least on trusted servers. At the same time, it's very likely that these same users will want to embrace the general, public cloud with its multitude of app-as-a-service options. Not surprisingly, there will be a need for isolating these workloads from interfering with each other.

Some examples of payloads that are better suited as traditional, local applications (and consequently within qubes), are MS Office for sensitive documents, large data-processing applications, and… networking and USB drivers and stacks. The latter things may not be very visible to the user, but we can't really offload them to the cloud. We have to host them on the local machine, and they present a huge attack surface that jeopardizes the user's other data and applications.

What about isolating web apps from each other, as well as protecting the host from them? Of course, that's the primary task of the Web browser. Yet, despite vendors' best efforts, browser security measures are still being circumvented. Continued expansion of the APIs that modern browsers expose to Web applications, such as WebGL, suggests that this state of affairs may not significantly improve in the foreseeable future.

What makes the Qubes model especially useful, I think, is that it allows us to put the whole browser in a container that is isolated by stronger mechanisms (simply because Qubes does not have to maintain all the interfaces that the browser must) and is managed by Qubes-defined policies. It's rather natural to imagine, e.g. a Chrome OS-based template for Qubes (perhaps even a unikernel-based one), from which lightweight browser VMs could be created, running either on the user's local machine, or in the cloud, as described above. Again, there will be pros and cons to both approaches, but Qubes should support both – and mostly seamlessly from the user's and admin's points of view (as well the Qubes service developer's point of view!).

Summary

Qubes Air is the next step on our roadmap to making the concept of "Security through Compartmentalization" applicable to more scenarios. It is also an attempt to address some of the biggest problems and weaknesses plaguing the current implementation of Qubes, specifically the difficulty of deployment and virtualization as a single point of failure. While Qubes-as-a-Service is one natural application that could be built on top of Qubes Air, it is certainly not the only one. We have also discussed running Qubes over clusters of physically isolated devices, as well as various hybrid scenarios. I believe the approach to security that Qubes has been implementing for years will continue to be valid for years to come, even in a world of apps-as-a-service.