**Part 3 [PAND, Container]**

July 24, 2023, 3:34PM

57m 37s

 **Mohit Gupta** 0:10  
Yes.  
Uh audio bhaskar.  
Buscar.  
Are you still there?  
We can't hear you.

 **Eduardo Toraya** joined the meeting

 **Vaithi Krishnasamy** joined the meeting

 **Bhaskar Verma** 0:28  
Uh, sorry, I was speaking on mute.

 **Dhiraj Singh** joined the meeting

 **Mohit Gupta** 0:30  
Mute.

 **Bhaskar Verma** 0:30  
Sorry.  
Bye bye.  
See. Yeah.  
OK.  
Hello, Ivan.  
I'll be covering pad medias is one of the update type that we published the media for.  
Just a quick recap, I covered APPF last week, so it's a very it's it's an updated that is very similar to APPF and I'll be discussing more on that how it is a bit similar to that.  
So bad media.  
I just said it.  
So Pan Media is basically pass native and it is very similar to APPF.  
It is used to run Azure related and other other teams.  
Uh services on on physical machines that are hosted in Azure data centers.  
So basically these are different parts of the same way like pass and native or two different things is just that it's they are part of the same package.  
So you can use it based on your requirement.  
The difference between the two is that native is the one that runs only Microsoft specific applications and pass the runs.  
What both Microsoft specific and external services that are that external customers basically deploy and tested on.  
So coming to the products for which we published this ban media is forward 2012.  
So our 2022RS1RS5 Windows 7 SP one and Windows 8.1.  
There are a bunch of products for which we create this PAN media image for ohm and yeah with.  
Can you go to the next slide?

 **Mohit Gupta** 2:06  
Setting.

 **Bhaskar Verma** 2:08  
Yeah.  
So As for the publishing for Pan Media, the first operation, like all the other update types, is the create operation won't be covering it in much detail because it is very same like very similar to to all the other updates.  
We only.  
WO creates a request that it sends one pub for getting the list of media that it sends that as a intent to SMF and then using that intent.  
Then, once that intent intent is signed off by SMF team that I'm media is generated, we go ahead with the other operations like ingest, publish, Go Live et cetera.  
So again, that's the next operation is the injection operation.  
In this operation, we don't do specifically much for PAND media.  
Do you think we do?  
Is that we create release proposal metadata documents for each file that we get as a payload from the SMF team.  
Whenever we hit the video Seeker API and and the copy of files happens in the next step, which is a published step.  
Can we go to the next slide?  
So publish operation for print media is the main operation.  
What we do over here is that we use this property EXE and we use the artifact store location information for each of the file that we get from the SMF team.

 **Richard Jimenez** joined the meeting

 **Bhaskar Verma** 3:27  
We copy all of these files to the Red Dog share and and that's pretty much it.  
In the publish operation that we do for bad media and yeah, the the next operation. Obviously is approve one we create a release. Uh approve release document and notify it every that the publisher. Primedia is done and once this is done.

 **Vaithi Krishnasamy** left the meeting

 **Vaithi Krishnasamy** joined the meeting

 **Bhaskar Verma** 3:48  
Then the HCC team, like there's no automated way, the in which HCC team knows that the published part for planning media is done is just it's a very manual process that we communicate to the HCC team that the deployment for PAND media, the copy of the replication to the Red Dog share has been done.  
And now you can start the deployment for the same images into different machines.  
So that's pretty much the entire flow for publishing for paired media.  
Uh, can we go to the next slide?  
Yeah.  
So As for the incidence, there are nothing there.  
No specific incidents for PAND media, but I just wanted to cover since we are.  
We are also covering the incidents, so these two incidents came up recently in 6B.  
Oh.  
Ohh Anton has a question ohh quick.

 **Achal Shah** 4:39  
They pass her.  
Yeah, quick question on you say post deploy post, uh approval there is they look at a AWE is it track it they look at or like what do they look at to start the time build.

 **Bhaskar Verma** 4:55  
I I remember I talked to one of the team members from the team and they said that they look at AWE place to check if the published part for Prime Media is done and based on the that they manually look at it and then start the development process. Yeah.

 **Achal Shah** 5:03  
OK, right.  
OK, there might be an opportunity to automate this, right?  
Because we I think we will, AWE does raise an event when publishing is complete so.

 **Bhaskar Verma** 5:25  
Yeah, correct.  
Correct.

 **Achal Shah** 5:26  
Yeah.

 **Bhaskar Verma** 5:26  
This is something that we can obviously automate and this is something that came also came in mind that we can probably improve this process of not manually looking at it or handing off to the HCC team as a manual process and probably do it automate in an automated fashion.  
So yeah.  
Umm yeah.  
On to the incident.  
So there, since we they're not, they're not specific incidents for pan Media, but since we are discussing the incidents, I just wanted to cover the issue incidents that came recently and they are also related to parent media images.  
So the first one is basically because of the PM migration work.  
So what happened was that the recently we moved the keyword that was in Microsoft tenant to the PME tenant and that those changes were not properly tested.  
Yeah, to the best possible way.  
And what happened was that once we started the ingestion operation and the publish operation, there was some issue there.  
We were not able to connect to the keyword and there were some also some issue related to keyword exchanges and they're not being installed in the new keyword.  
So that was one of the incident.  
Because of that and the other incident was a very similar incident where we made some changes but could not test you.  
So second one was related to service bus.  
This incident was when Alexey has a question I guess.

 **Alexey Loginov** 6:54  
Yeah, I I can wait.

 **Bhaskar Verma** 6:56  
OK.  
So second incident, I just want to quickly cover.  
So it is regarding the content operation dispatcher logic app not working properly.  
So there was some issue in the service bus and we had to create a new service bus to let the request pass.  
So this something like I think there are a lot of requests that was being sent to the service bus and some of them got lost and that was the incident that came because of that.  
And we created a new service extension.  
I think polo can give a more detailed definition because he did the mitigation for it.  
But yeah, the that's the detail on these two incidents.  
And he, Alicia, other question I guess.

 **Alexey Loginov** 7:42  
Yeah, I have a generic question.  
I I was looking at yesterday at last time update types I prepare and now this band and like and and trying to find a good mind map for myself was the difference.  
Both of them used by Asia team for specific scenarios.  
I get that.  
But you know from publishing perspective, they should be more or less the same.  
Trying to have a good understanding in my head why the flow from publishing perspective is different.  
Can somebody explain to me what's what?

 **Bhaskar Verma** 8:14  
E yeah.

 **Alexey Loginov** 8:15  
What's that like with?  
Maybe you have some side by side?  
I would expect to have pretty much the same flow for those.  
Why?  
Why is different?

 **Mohit Gupta** 8:23  
Cool.

 **Bhaskar Verma** 8:24  
Yeah.

 **Mohit Gupta** 8:25  
That's a good question and good.

 **Bhaskar Verma** 8:25  
So Alexey to answer a question, sorry, more go ahead.

 **Mohit Gupta** 8:30  
So generally speaking, it it depends on what our end customers are expecting.  
So for example, deployment team expects a different notification for APPF.  
They expect the metadata file AP signature DOT CSV in pan.  
That is not the case.  
If I'm right faster.

 **Bhaskar Verma** 8:50  
Uh, yes.

 **Mohit Gupta** 8:50  
Uh, that that kind of drives what we do differently in the publishing pipeline and similarly when we go to other update types, the end location, the, the place where we publish to is different for containers, it's registry for ISO, HSRP for VSTS, just partner central for ESD, it is uh, I've just says Rd cat.  
So those are the things that actually drive what we do in, in the publishing pipeline, in media publishing.

 **Alexey Loginov** 9:25  
And deployment team is the team who takes it, takes the media that we published and do whatever this team has to do.

 **Mohit Gupta** 9:34  
We only only for APS bank, the deployment team.  
Yeah, the deployment team works only for a PF and timed media.

 **Alexey Loginov** 9:44  
OK.  
And the the two deployment teams for different update types and then expect input for them output from us in a different format and that's why we have different flow.

 **Mohit Gupta** 9:57  
I think the format is the same but the the way we notify them is different for APPF they expect a metadata file to be in the folder before they start their end of publishing.  
For panned, it's I think this notification which Bhaskar just talked about, which can be probably automated.

 **Alexey Loginov** 10:23  
Thank you.

 **Mohit Gupta** 10:30  
Anything anybody wants to add before we move to the on demand API for panned?

 **Dhiraj Singh** 10:37  
Uh, so the way to think about the two pipelines we are presented till now is both delivered to Red Dog.  
The publishing pipeline is relatively robust.  
We had had few incidents but those were really more of infrastructure incidents, so nothing to do here.  
That's my understanding till now.

 **Mohit Gupta** 10:59  
More or less, yes.  
There's one class of incidents we will discuss when we finish all the update types is in case the metadata file is not present for APPF.

 **Dhiraj Singh** 11:01  
And.

 **Mohit Gupta** 11:11  
Then, instead of rerunning the whole logic app, can we just run the generation of that APPF?  
Sorry, AP signature dot CSV file and that's all.  
We don't have to copy the GPS of data again.  
So can we break down the current service into two steps?  
Umm.  
And do it step wise.  
That's the only.

 **Dhiraj Singh** 11:32  
I the so.

 **Mohit Gupta** 11:32  
That's my understand.  
Anybody has any other ideas?

 **Dhiraj Singh** 11:36  
Yeah, I will request then if we can present the incident in the context while we are discussing.  
Otherwise, we'll forget, right?  
So if we're discussing PAND and if we, yeah.

 **Mohit Gupta** 11:44  
Yeah.  
So these incidents are in not not that context.  
So these incidents do not warrant any design change or any and any other change.  
I was talking from Ristic point of view for all for both APPF and banned.

 **Dhiraj Singh** 11:56  
Could.

 **Achal Shah** 11:56  
There were no yeah.  
They are not specific to those updates, right?  
I mean there are infra issues and.

 **Mohit Gupta** 12:08  
2 Yep. Yep.  
So yeah, at a high level, yes, you're right.  
There are no issues identified till now, at least in a UPS which require any change.  
Yes, waiting.

 **Vaithi Krishnasamy** 12:27  
So so here you are saying about, you know, like when when something fails.  
We did.  
We do retry the entire logic app you know to to mitigate that, right?  
So, so so being like worked on media before and and now on the UUP, right.  
So one good thing I see on the UPS side is how the retries are handled right, which I think you know we can optimize on the media side.  
So how currently it works on the media side at least, is sorry, at least on the UUP side is whenever there is a failure that particular action that particular step on the logic app you know goes on the waiting state rather than going on an error occurred state right.  
So what happens is like, you know, like I think it was an intern, right, Derek who built a retry UI like a DevOps UI and everywhere where that incident gets registered on a retry UI where entry in the database is created with the callback URL just for that logic app step right?  
So that logic app doesn't error occur.  
Uh, but the entry is created with the call back to that step, right?  
So it it is just going to be waiting instead of getting you know error occurred and then you know you go fix whatever you wanna do and then you know just click the retry button so it's going to do whatever it has to do just for that step and then it's going to make a callback to the logic app exactly for that step right.  
So it stops retrying the whole logic app right.  
I think you know that is something we probably be a good investment.

 **Mohit Gupta** 14:00  
Yeah.  
So.  
So there's a there's a different thread going on for that already.  
Uh.  
Arvind is working on a better, smarter retries for Media Publishing as well.  
And I think Viraj you spoke about the same thing, right?  
They retry manager or right?

 **Dhiraj Singh** 14:15  
Yeah, I've requested or went to connect with Nabajit on retry design.  
I think that helps overall, but I'm still saying that for PAND and the other pipeline, we are saying even there retry is not a problem.

 **Aravind Siddoju** 14:22  
Uh.

 **Dhiraj Singh** 14:31  
It just works right?  
I think we have not heard significant issues.  
So I don't want to tie the two things together.

 **Mohit Gupta** 14:35  
Yeah.

 **Dhiraj Singh** 14:37  
I want, I think, if we can just go pipeline by pipeline and say, hey, this pipeline has some serious issues, then I think we can just focus on and drill down there.  
So why do you write for retry or wind presented a plan and which looks similar to retry.

 **Mohit Gupta** 14:48  
Sure, if it's a.

 **Dhiraj Singh** 14:57  
So I have requested him to connect with Nabajit and see if they are synergies and we can reuse retry manager and even if we are not using maybe we can borrow some new concepts into try manager and vice versa.

 **Vaithi Krishnasamy** 15:05  
Yeah.  
Yeah, I mean like before I I I forget like like one more thing as we are talking about this is how the how the incidents are getting generated right again I I do not know the current state of media but like few years back the incidents were plugged into the logic apps, right?  
So any times an incident is get generated, the incidents are not reached, right?  
It doesn't have a lot of lot of like data on the incidence right?  
So it's a very generic incidence which gets raised and it took a lot of developers time to go figure out, you know, like what failed, you know, like and everything, right, rather than everything getting populated in an incident.  
So the one suggestion is you know, removing the incident generation from a logic app and and moving it to a code part.  
We know it would also save some developer time.

 **Aravind Siddoju** 15:58  
So why the uh, just you bought that incident part.  
So right now the media services are filing incidents at the service level as well as at the logic capsule.  
We kind of have that detailed exception, not fully detailed, but that exception level incident and then we have a larger cap level incident getting file which is something that we want to umm get rid of with the retry thing that I'm trying out.

 **Vaithi Krishnasamy** 16:17  
Umm.  
What?

 **Aravind Siddoju** 16:28  
But yeah, for every service we have this logic app filing one incident.

 **Vaithi Krishnasamy** 16:28  
Thanks.  
Thanks everybody.

 **Aravind Siddoju** 16:32  
I had the service level.  
We file one more.

 **Vaithi Krishnasamy** 16:34  
God.  
OK.  
Makes sense.  
Thanks aravind.

 **Mohit Gupta** 16:41  
Moving on to the on Demand publishing API, I've been over to you.

 **Aravind Siddoju** 16:48  
Uh.  
All right, I'll.  
I'll just give little requirements for this new API that we have created four months ago.  
It all started with SMF looking for a simple and faster pipeline to reach out to the one P teams.  
So whatever, just now, Basker has discussed about a PF and PAND where the customers are basically the binge and office team and they often come up with a customizations of their media, today's MF team.  
So SMF, what they wanted to achieve is they they wanted to reach out to this one P teams faster for validating the customizations they made.  
This is to ensure that good quality of images that we published during the builds, as well as no regressions during the release.  
So.  
So they are looking for a pipeline or a simpler way where they can quickly reach out to the one P team.  
So that's where it is.  
New API that we have generated so alerts hiding shows as test APPF pan.  
The initial requirement is for APPF publishing of customized media for friend to the the 1P teams.  
So that's, that's where we built an API without AWE coming into picture.  
So it would be a simple handshake between SMF team and the the one pub which in fact the media team itself.  
So SMF can just request us to publish set of images for that particular product and particular media and the one pub API takes care of publishing it directly to the one P teams without that orchestrated flow where it takes some time in order to orchestrate operation by operation like create in just publish against can go live and approve and golive.  
So those are are bypassed.  
This API basically just gets the request and eventually tries to do the the ultimate operational publishing into the the 1P team.  
So that's that's this API is about be we started with APPF and once it became more useful for them and they've been using it uh for time for some time now it's been 2-3 months.  
They've been using this API vigorously to publish this customized area, so eventually the pan also started requesting us to enhance this for publishing the pan media as well.  
So we we tried integrating that as well using this API.  
Umm, so that's that's about the the test API for Media publishing, where we really don't need AWE to publish the media rather.  
Uh.  
Handshake between SMF and the one pub which takes care of publishing it again.  
It's we are not going to break any production architected things, but it said different way of publishing where you really need AWE where you really don't need that release based information to reach out to the one P teams.  
Umm, that's that's about the the test API Viraj.

 **Dhiraj Singh** 20:12  
Yeah.  
How do we count?  
This is a test API, right?  
So we are allowing my understanding is we are allowing a third party to like inject ohh publishing into the system.

 **Nabajit Pathak** joined the meeting

 **Dhiraj Singh** 20:24  
How do we control that this doesn't end up in production systems and all?  
Like what are the gates and controls?

 **Aravind Siddoju** 20:30  
So, right, we we don't even push our like publish anything to the production endpoint although like the normal production endpoints that we do in our regular flow publishing.  
So this goes to a different location where the partner teams can differentiate it out between the actual production based media and the customized media that without.

 **Dhiraj Singh** 20:56  
So anything via test API has some different identifier.  
It's always guaranteed to go to some different endpoint.

 **Aravind Siddoju** 21:05  
Right, right if.

 **Dhiraj Singh** 21:05  
It's not.

 **Aravind Siddoju** 21:12  
Uh, so?

 **Alexey Loginov** 21:12  
And who is the main customer or in Aravind of this API who uses it?

 **Aravind Siddoju** 21:17  
Uh, so right now the APPF and panned is something which is using it.  
That would be again the 1P teams of binge office and everything who are trying to pull this images from that deployment pipeline.  
The customers for this API.  
Umm any other questions?  
So it's again maybe of a Mohit, do you want me to give that diagram feature is better.

 **Mohit Gupta** 21:50  
No.  
So.  
So it's a slight detour, but I thought it was.  
Relevant because we were talking about and so we can have a a dedicated 15 minute slot for this new handler that you created for on demand publishing.  
But uh, I do not think we should go into it today, but we can go go into it.  
If if there's a, if there's a need in future.

 **Aravind Siddoju** 22:19  
Yeah.

 **Mohit Gupta** 22:19  
We could we could move to container now if no one has any questions on this.

 **Achal Shah** 22:30  
And just to be clear on the API, it's they call uh one API directly and there's no logic app involvement either, right?

 **Aravind Siddoju** 22:30  
Umm.  
Right.

 **Mohit Gupta** 22:45  
You.  
Happy.

 **Aravind Siddoju** 22:52  
Uh, alright, so I'll I'll cover the overview part of containers and what it does and how we publish and what are you gets involved in the publishing of containers.  
Then maybe we can talk about the foreign layer removal thing that check up recently did.  
Yeah, this first two slides are just about the interaction to containers.  
I just not to do a complete presentation, but yeah, again umm the container images again, the container world and the darker world.  
The images are like blue prints that can be stored.  
Images can be the OS image or an application image on top of other base images and they are like a passive instance of whatever you want to run as a container.  
So that again defines the definition of container, which is basically running instances of the images that we publish.  
The darker world and but as far as the container media publishing is concerned, we we publish only Windows OS images to the customers and we we store these images in the container registries.  
The typical container registries are Docker hub and Azure container registries, which is an Azure offering where you can create your own private Azure registries.  
Uh, how much can you go to the next slide?  
Alright, so MCR, which stands for Microsoft Container Registry, is a final registered destination for our container images.  
This is the place where our the images that we get from SMF will eventually get pushed to the way it works is we don't really publish to the MCR, rather we publish to our own private Azure Container registry named Unpub ACR Prod.  
This is the the registry that the one pub team owns and handles it and we do have a replication system onboarded and hook to MCR whenever we copy or publish anything to the one pub ACR prod registry gets replicated to the MCR.  
MCR is so that's.  
That's how the the publishing happens to the the final Microsoft Container registry.  
Umm Mohit next flight?  
So I sorry, sorry for the blood image.  
I was.  
I didn't have the time to capture the umm and from the portal so, but this is how a basic.  
Container registry looks like this is our R1 pub ACR container registry.  
If you just look at those layers, which is the public's followed by slash followed by Windows, the middle ones, so they are all kind of folders within the registry which we call them as repositories, and the images will get stored in the final repository in that path.  
That's where we actually store the images.  
This is how no the registry structure will look like on the Azure and if you just look closely, we do have a few prefixes with internal and a few few sorry repository.  
Starting with prefixes public.  
Anything where we push our we publish images to this internal prefix.  
Repositories gets copied or gets replicated to a registry named Msint, which is a registry for internal publishing where the no internal clients can pull the images and test it out and the anything that we push our publish to the the public the repository starting with the public.  
Will get actually replicated to the MCR registry.  
That's how the replication works, and that's how the no the images gets published to either the internal registry or the the MCR registry.  
Mohit, can you go to next slide please?

 **Venkatnagaraju Goursetti** joined the meeting

 **Aravind Siddoju** 26:52  
Uh, again, the slide is about the the internal publishing.  
As I just mentioned, anything that's starting with the prefix of internal within our ACR goes to a registry named Msint.  
Again, this is just only available to our internal employees.  
Within Microsoft, we published the preleased images to this mess int and we don't have to do anything.  
There is specific rather we push our publish the images into that internal repository folder and the replication is onboarded to replicate it to the Ms INT registry and that's where the the customers or the internal customers can pull the images and try it out.  
That's about the internal publishing Mohit.  
Next slide please.  
Just want to bring this picture.  
I pulled it from one of our container.  
Slides.  
So this is how it looks from the AWE forefront and what happens on the the the container publishing side.  
So media creation is again just a intent.  
Yeah, intent notification sent to the media and once the intent media creation is completed, so that's when AWE kicks off.  
The second thing which is ingestion.  
Umm.  
The intuition is a place where we actually do a lot of operation for the the containers.  
I'll I'll talk about that in the next slide.  
But this is the the second layer is a place where we actually upload or we pull the the container information from the SMS and we upload it to our one pub ACR and we we upload it into the BLOB storage the layers and eventually we push the the container media into our one pub PCR prod and then we have this uh publish the didn't the publish.  
We actually just mount our replicate the images from in just folder to the internal folder.  
So that's when.  
They get replicated to Ms internal prod and that's sorry Ms in registry and that's when they are available for the internal customers for trying it out and the 4th one is again RM approval and then eventually the notify where we actually publish the images to the MCR prod and they'll be available to the end customers and in the next lesson we talk about like what exactly happens in this each of the steps.  
But these are the the five stages that happens.  
Mohit, if you just go to the next slide.  
Ohh sorry Achal.  
No, go ahead.

 **Achal Shah** 29:35  
Yeah, quick question.

 **Aravind Siddoju** 29:41  
Umm.  
Ah, no.  
So no.

 **Achal Shah** 29:48  
No. OK.

 **Aravind Siddoju** 29:50  
So during the injection itself we so we basically upload the the container.  
Information into the blobs, MCR, BLOB storage, and eventually we again ingest we we published the even that information into one pub Asia prior at the same time, but we put that in the ingest folder within one of HCR prod uh hope that that answers your question.  
So it it just stays in that in just folder, but it doesn't really become public yet, so the ingest would be again replicated in the MCR.  
Fraud ingest.  
Umm, repository within it, but it's not available for the public yet.

 **Achal Shah** 30:32  
Yeah, I was just trying to see what the distinction between ingestion and publishing is.

 **Aravind Siddoju** 30:36  
I'll.  
I'll I'll show you in a minute.

 **Achal Shah** 30:37  
Yeah.  
OK.  
OK, cool. Thanks.

 **Aravind Siddoju** 30:40  
Like so, this is the the core part of the container publishing where the ingestion happens.  
Ingestion again is all about.  
We extract pretty much all the information from the SMF team from the drop ideas that they provide in terms of this media seeker URLs.  
So we typically get the second list of information like the manifest file related to that container manifest list, that JSON file related to container can't pick which basically had contains the metadata for that container and tap that geez is the actual image of that we want to publish and then we have this parent layer info if at all.  
If it's a delta layer, it it has its base layer which we call the parent layer information as well.  
So these we get this pretty much all the information from the props that SMF team provides and we during the engine itself, we upload entire information to MCR BLOB storage.  
Once we have everything on MCR BLOB, uh, we we eventually push the manifest layers into the ingest repository within one pub ACR prod.  
So pretty much I'll upload happens in the ingest and we just I just find it to the ingest folder within month of ACR proc and most of the the business are the processing.  
Everything happens in the engine itself.  
Now if you just go to the next slide, these are all just grass mounting the ingestion folder ingestion itself.  
So for the container published, we just cross mount the repository to internal as soon as we cross mode in month of ACR prod, the replication happens to replicate the same image information into Ms internal.  
But did you see that's where it becomes available to the internal audience for testing the?  
Again, the publishing is just about the class mounting and it's making sure it is available in the MSDN for the internal customers.  
We don't do really much in the publish if you just go to the next slide Mohit, then we have the approval step.  
Uh, where we again from the one pub ACR prod registry itself.  
We crossmon the layers from the internal repository to repository, starting with staging on the one pub ACR prod.  
Now again the same thing gets replicated in the MCR prod registry, which is the actual registry that MCR team gets.  
Uh makes them visible to the final customers, although they are not publicly available yet.  
But we we kind of put it in a repository name staging so that pretty much everything is ready in the MCR registry itself at the approve action itself and more if you just go to the next step.  
So this is about go life.  
We just don't do much.  
I can just cross mount from staging to the public repository.  
That's when again the replication happens on the the MCR side where the the images gets replicated in this public repository.  
That's when MCR makes these images publicly exposed.  
Umm, so if you just look closely it's it's more on the injection where we actually do all this business logic and kind of process all the the media information, the image information, it's configs, manifest files, parent layer information, we upload everything to the BLOB and eventually we push that manifest files to the one pub ACR prod in just repository within it and the later stages are more about just grass mounting that through a different repository and then uh in the background the replication happens.  
Uh, which is kind of already automated and we we don't need to do much.  
It should just like a one time set process, which we do in the beginning when we set up the one per base product and that eventually when the golive happens, that's when they moved to the public on the MCR registry.  
That's when they become available to the final audience.  
Production audience, that's how.  
That's how the container flow works.  
Umm, uh, Jacob, feel free to add if I missed any and.

 **Jacob Kissel** 35:05  
Yeah, there's one more part of Go live, which is that we maintain a GitHub.  
Or we don't maintain it.  
MCR maintains a a GitHub repo and that we commit to automatically for go live, which keeps the container versions and KB numbers for like the tag names of the containers in there, and MCR uses that GitHub as like data for a kind of front end discovery portal that they that you can browse to through Docker hub.  
So you can go to the Docker hub page and see what's the latest tag for Nano Server, server core, or whatever client enterprise, and we update that at the end of go live as well to show this is the latest KB number tag.  
This is the latest LCU version tag, so that's the that's the last step.  
That's like the very, very last step of go live after we're done with everything.  
Umm.  
And then also, uh, there is one more, which is that now we're signing the container manifests.  
So we we're publishing the images, but we're also publishing signed manifests of them.  
So customers can verify once the images on their machine if the digest matches what the Microsoft Cert signed manifest says it should be to know that it's authentic or not.

 **Achal Shah** 36:27  
Is that signing done during ingestion or public later on?

 **Jacob Kissel** 36:32  
So the signing is done for the internal published images and the staged and go live images.  
It's not done during ingestion.  
Umm.  
But basically, the signature, uh, the signing and the publishing of the sign to manifest happens during the publish.  
Any of the type of publish steps, whether it's internal or public.

 **Achal Shah** 36:57  
And during ingestion we copy stuff to the MCR BLOB store.  
What?  
What's that for?

 **Jacob Kissel** 37:07  
So we don't we copy stuff to a BLOB storage that we own and we copy stuff to the ingestion registry of our ACR, but nothing actually makes it all the way into the MCR.  
Excuse me?  
Systems until publishing and go live.

 **Achal Shah** 37:20  
OK. OK.  
So that BLOB story is is probably ours.

 **Vaithi Krishnasamy** 37:24  
So I I just, I just like I can give some history on that one, right.

 **Jacob Kissel** 37:25  
Yeah.

 **Achal Shah** 37:25  
Yeah, yeah.

 **Vaithi Krishnasamy** 37:28  
So it was copied to a BLOB store at that point, because you know previously when container publishing first started right, the containers were produced in a in a file share right in a file storage.  
So not somewhere, not not on a cloud, right?  
So.  
So what we wanted to do is, you know, we wanted it to bring it to a cloud so that we can do like a cloud to cloud copy.  
When you take from a BLOB storage uh and put it on an ACR, it is the copies a lot faster rather than directly, you know, streaming from a file storage at the end moment you know where you you there is a good chance that the copy would fail, right?  
So that's why pre copied to a BLOB storage.  
So at a later date you can do a BLOB to BLOB cloud to cloud copy, but this was during the time frame when things were generated on a file share, right?  
I'm not sure if it is still relevant because with PME right I don't think that step is would be necessary as my is my assumption yeah.

 **Jacob Kissel** 38:27  
While we still do the BLOB storage upload for ESRP submission as well.

 **Achal Shah** 38:28  
Yeah, Mike.

 **Vaithi Krishnasamy** 38:36  
OK, ESRP OK.

 **Dhiraj Singh** 38:39  
Yeah, but uh, I think there was an effort to remove the redundant file copies a long time back, and actually we asserted that, yeah, we are doing minimal file copies, which are really needed.  
So I think in this and that really simplifies and all also add cost advantages, right?  
Because then you are saying I'm just taking.  
Maybe I'm reconfiguring and rejigging the metadata required for publishing, but the file remains at source and we'll do a call direct copy from the source the target.  
So if possible, I think we should look here if we can.  
I think publishing will become simpler.  
We move to that model.

 **Jacob Kissel** 39:21  
Yeah, that's a good option.

 **Alexey Loginov** 39:25  
Can I quickly also ask?  
I don't want to digress, but since we are doing cloud publishing for containers, what's the main reason we are still using Red Dog shares for the update types that we just discussed?  
Why they they end up on the share, not so in the cloud.  
Is it size issue or it just legacy?

 **Jacob Kissel** 39:47  
Are you talking about Autopilot and past native?

 **Alexey Loginov** 39:50  
Yes.

 **Mohit Gupta** 39:50  
Yes.

 **Jacob Kissel** 39:51  
Yes, so, uh, for those ones, it's about the restriction of the consumers.  
What it boils down to Autopilot is actually in the middle of getting migrated off of Red Dog.  
Red Dog is kind of just an implementation detail of publishing into Autopilot, and we'll soon be using an ADO tool that the Autopilot team developed called Stratus, which lets us deploy into autopilot without using Red Dog.  
But for past native, the consumers of these images, which is the past native deployment team that sends the images we give them, they basically control the rollout across all of their ecosystem.  
They use a tool called easy deployer and the tool that that team uses is very tightly coupled to Red Dog specifically for like some historical reasons, and it's not even just coupled to like getting it from some Corp net file share.  
They they've told us that literally like the images have to be on Red Dog specifically for them to be able to deploy them.  
So we've been talking to them and trying to see if there's a way for us to remove that coupling, but unfortunately, uh, as it stands right now, they're not able to do that.  
And we're trying to get the work scheduled as soon as we can, but it it's looking like for them it's it's a lower priority and it seems like it's kind of a very, very historical tech debt on their side that's going to take a significant amount of effort to tackle.

 **Mohit Gupta** 41:10  
Yeah.

 **Jacob Kissel** 41:20  
So we're stuck right now.  
We have to publish to Red Dog for that specific coupling for past native, but yeah, like for APPF we're very happy that we can be moving off of that very soon.  
Umm but originally that was there because it was an implementation detail like the APPF deployment system.  
Like it's different from the past native one, the app deployment system.  
Red Dog was one of the options, but they have the Stratus option as well, which doesn't use red dogs, so we're moving too Stratus.  
But past native, there's no other options right now.

 **Mohit Gupta** 41:52  
I.

 **Jacob Kissel** 41:55  
It's just Red Dog because of the coupling that they have.

 **Alexey Loginov** 42:00  
Yeah.  
Jacob makes sense.  
I kind of expected it, but it's one thing for them to change their deployment strategy.  
It's another thing to, I don't know, publish it somewhere in the cloud and send them notification, and if they want to copy to reddog, they can do it as part of their deployment process.  
Like we can meet somewhere in the middle, right?

 **Jacob Kissel** 42:23  
Yeah.

 **Alexey Loginov** 42:24  
It could be they still need a dog.  
Here is your cloud location.

 **Jacob Kissel** 42:29  
Right.

 **Alexey Loginov** 42:29  
Copy it.  
It could be somewhere in the middle, right?

 **Jacob Kissel** 42:31  
Yeah.  
Yeah.  
No, I totally agree.  
So we we want we have a I talked to Fabi last Friday and we're planning on trying to get a learning session with them to learn what all of their couplings look like and how their system set up at the entry point.  
So like what happens when they take our files and and where do they do?  
What do they do with them after that?  
And if they, I think it's one thing if they can say yeah in the next quarter we can move off of Red Dog entirely, then we won't have to make this in between work.  
But if they if they can't give like a definite you know timeline. I think it's a great idea for us to suggest. Hey, what, if we just created this middle service that that the new contract is we give it to you in the cloud and and from there, you take it and put it where it needs to be, because then when they move off of red dog.  
They can just take it from the cloud and put it wherever else they need to put it.  
So this is definitely idea that we want to pursue with them because especially if they can't really give us a solid time frame for migrating off of Red Dog, it's for sure something that we don't want to maintain for that long either.  
So definitely agree on that side.

 **Fabiola Ansara** 43:44  
Yeah, we were discussing exactly that.  
And yeah, the intention is to understand how the band work like in their system.  
So we can better help them.  
So yeah, I was going to do exactly that.  
Yeah.  
Thanks for bringing that up.

 **Mohit Gupta** 44:18  
So that was pretty much about container.  
We have about 10 minutes.  
Jacob, if you want to talk about the cool stuff that you did with the the layers, the foreign layers.  
The parent layers.

 **Jacob Kissel** 44:33  
Yeah, for sure.  
Yeah.  
So I'll go ahead.

 **Dhiraj Singh** 44:34  
Good.  
Uh, before we go like what are the issues in container like incidents we didn't talk about here.

 **Jacob Kissel** 44:45  
Or do we have a list of incidents go through?

 **Mohit Gupta** 44:45  
So.  
So we we have a list of incidents, but I think we'll go over it once we have a good understanding end to end off and one update type for APPF.  
Uh.  
Uh basket.  
Brent brought the incidents, but what we are planning is we'll discuss incidents and the categories of incidents in the end once we are done with all update types said.

 **Dhiraj Singh** 45:09  
I see. OK.

 **Fabiola Ansara** 45:13  
Yeah.  
One of the incidents that we like, we can talk about is a part of the.  
Areas one parent layer, but actually Jacob is going to talk about the friendly removal, but one of the incidents was that it was too big.  
It was around like 3.5 gigs and we were not expected that size, and Jacob can definitely.

 **Jacob Kissel** 45:30  
Yeah.

 **Fabiola Ansara** 45:34  
Talk about this.  
In the time he is talking about the foreign layer removal, so go ahead Jacob.

 **Jacob Kissel** 45:38  
Yeah.  
Yeah, for sure.  
So there there's kind of three overall topics here.  
One is the uh or it's two?  
There's one is the foreign layer and one is the automated rebase line.  
Umm, they happen to happen in the same month because the base container team kind of said a rebase line for anyone that's unfamiliar with container specific.  
It's very similar to LCU and other types of rebaseline where normally on a month we have one parent layer which came out at the RTM for that product and then 1D layer which is only having the LC for that month data.  
The parent layer is the same month to month and the LCD is new every month.  
The delta layer.  
Sorry.  
So on a rebaseline event, you squash those together into a brand new parent layer that will be the same from that month going forward until the next rebase line event.  
So umm, since the rebase line is kind of like a reset soft reset whatever you wanna call it for containers, the base container team kind of said since we're doing the foreign layer removal, which is another kind of big change to containers, let's combine them into the same month and give kind of all the good news to customers at the same time kind of the rebaseline.  
We'll also be a good kind of new starting line for the foreign layer removed images.  
So that's the rebaseline thing.  
The overview of foreign layers is that previously, if I was a customer who wanted to build like my own application on top of a Windows image, I would be able to serve my customers my application related bits, but I would not be able to serve them windows related layers.  
So the customer pulling my image would be getting windows layers from the MCR, which our company owns, and then my applications from the other.  
Like whatever my own registry is.  
If I were serving a Linux image with my application on top, I could actually give my customers my Linux like the actual Linux operating system layers and my applications, all from my own registry.  
So that's kind of the overall idea of what foreign layers were basically foreign layers made it so that only Microsoft was the one serving the layers out from our container registry at all times.  
Uh, the overall reason why we want to move away from them is that first of all, Windows is the only one that was ever using this foreign layer technology and containers.  
So removing it means that we line up with the rest of the ecosystem, like let it all the Linux containers don't use foreign layers.  
Secondly, is the second biggest reason is air gap.  
Clouds don't play nice with foreign layers because if you're air gap or your environment that has a security rules on it allows traffic to your registry, you can get all your images from there.  
But then as soon as you can't find the windows layer in there, you have to go find it from the Microsoft Container registry.  
You'll have to set up another rule for that, so some air gapped clouds just wouldn't allow you to add that rule.  
It's everything is depending on their own companies like security rules, but then even like the the US Nat and US SEC.  
Ergot clouds that Microsoft owns this the foreign layers that pointed to MCR.  
MCR.  
That public URL, that's for sure not able to leave the air gap cloud boundaries.  
So they had to do work around so that these foreign layers would be served directly from registries in the air gap cloud.  
So kind of the summary is those are the two biggest reasons why we wanted to remove foreign layers and in the same month basically now from 6B moving forward where publishing images that are not four and they're just normal layers.  
So if I pull an image, I build an application on top of it and I push it into my own registry.  
I'll now be able to push the Windows image into my registry and I can kind of give it out to my customers and everything.  
Obviously the caveat is you can't run a Windows container on anything besides a Windows operating system, so that's why it's kind of they figured it was OK for this because it's not like we can give Windows containers away for free.  
You're not going to be able to use it unless you're paying for a Windows OS anyways, so that was kind of what made it OK for this change, but ultimately the removal of foreign layers was as simple as the manifest we publish, which kind of describes the image, just doesn't say foreign layer on it anymore.  
Just says that it's a normal layer O that happened in the same month as rebaselining and in this month in the past we've rebase lined RS5 and server 2022, which means we've squashed those in the past, sometimes for size related reasons, and one of the times it was actually because there's a specific type, a few specific things that you can't change in the parent layer through the use of a delta layer diff.  
So we actually had to rebaseline because we needed to change something in the parent layer.  
UM, but now another reason would be like if the delta layer got too big, you can do one month of rebaselining, squash everything into a new parent layer, and the next month delta layer will be significantly smaller.  
So for a number of reasons, we decided to rebaseline at the same time as we're moving forward in layers and it's the first time we've actually rebaseline Rs one since, uh.  
I personally don't remember.  
We definitely haven't rebaseline Rs one since 2019 when I joined the company.  
I don't know if it was rebaseline before that, but we weren't really publishing containers for that long before that, so it's most likely the first time we've rebased lined ours one ever and ours one layers are the biggest of all the containers because it's the first container we started publishing.  
It's got the most kind of inefficient usage of all the containers.  
Like fancy things you can do with containers and lowering the size of them.  
So when we rebase line the one containers, the new parent layer with all the extra delta layers added in is much bigger than any of the delta layers we were used to publishing through our system before.  
Umm, so our system got kind of clogged up.  
The parallelism wasn't like kind of ready to ingest that much data.  
At the same time, and additionally our as one we publish 17 different languages when we originally started publishing RS1 containers, they wanted to publish one for each of the localizations.  
But for R, all the other containers moving forward, we don't publish all the different localizations, it's just Nano server, server, core server, client enterprise for the different products.  
But for Rs one, it's only server core, but it's one per 17 languages.  
So not only are the layers bigger, but there's also much more of them.  
So we hit a issue with our code which is getting really really stopped up.  
It was not able to.

 **Achal Shah** left the meeting

 **Jacob Kissel** 52:16  
Uh.  
Ingest all of these huge pieces of data at the same time, so we had to increase or we had to improve the parallelization code to make it be able to handle that.  
So that was one of the big incidents that month.  
And then there was another incident related to uh, now that we're pre rebaseline has always been a concept in containers.  
We've done it before 6B, but it's just been manual in the past where we have to actually commit a change to our code.  
What for one month and then remove that change after the rebaseline month because our code is expecting a certain layer to be one or two layers at any point in time, but the code change here is that we are automatically able to rebase or accept rebase line images into.  
Are publishing system at any point in time.  
So if we get one layer or two layers from HCC for an image, we can publish it either way and it's based on a type of file they give us called parent layer info.  
If that file exists, then we're able to go find the parent layer that they made in the past.  
If it doesn't exist, the assumption is that it's a rebase lined image, and there should only be one layer.  
So one of the other issues was with the parent layer info.  
JSON wasn't getting generated for one of the images and then there was another one where, uh, we were.  
It was getting generated but one of our types of ingestions was failing to actually ingest it.  
So basically we had a delta layer and we have checks and balances throughout our system.  
So it got to a piece of ingestion that saw the manifest and it said, hey, this manifest has two layers in it.  
We're expecting apparent layer to be somewhere, but we didn't find apparent layer from the HTC drop that would they were they gave us, so it raised an ICM saying hey, we're looking for a parent layer, but we didn't find one and we transferred it to HCC because something similar has happened in the past.  
But it turned out that HCC was publishing it and our system was just failing to engg.  
Test it so at that point we kind of fixed the bug that was missing the parent layer that time.  
So there were a number of kind of learnings throughout this automatic rebase lining system that we added in, but it worked pretty well for the last publishing of 7B.  
So.  
Umm.  
Hopefully we're in the clear now, but those are two of the main incidents and those are the two new kind of container related work for the removal of foreign layers as well as the automatic rebase lining for publishing.

 **Alexey Loginov** left the meeting

 **Jacob Kissel** 54:50  
Does anyone have any questions about those or the incidents or anything like that?  
Either it means no questions or I was talking too fast without any slides in the background.  
So sorry, maybe hard to follow my rambling, but I'm also gonna be doing a I think in the this month UPS monthly I have a presentation as well about this this stuff related.  
So, umm, that'd be good as well.

 **Mohit Gupta** 55:28  
Yes.  
Yeah, that would be great.

 **Venkatnagaraju Goursetti** left the meeting

 **Mohit Gupta** 55:35  
Thanks Jacob.  
So I think there are almost on time and looks like we don't need a follow up for containers.  
So next week we'll be talking about Ohh get HD's and that's going to be interesting with I think polo taking the lead for all his interactions with the partner center team and all the things that we do for VHD's.  
VHD's.  
'S I think that will be one of the most involved update types.  
If we have time, we can also accommodate uh ISO along with the C.  
Any questions?  
Anything else you want to hear in the next session?

 **Jacob Kissel** 56:35  
If this specific audience wants to hear about the way we're moving off of Red Dog for Autopilot, I could bring a couple of slides about that.  
That work is like in progress right now and I'm I'm basically done with the proof of concept.  
It's just not in our code, but if you if you guys want to hear about that, I can prepare something for that for the next session.

 **Mohit Gupta** 56:54  
That would be great.  
Jacob uh, so you need 5 minutes, 10 minutes.

 **Jacob Kissel** 57:00  
Uh, yeah, 5 minutes should be fine.

 **Mohit Gupta** 57:03  
OK, OK.

 **Jacob Kissel** 57:03  
It's.

 **Mohit Gupta** 57:03  
So I'll include it.

 **Jacob Kissel** 57:04  
Yeah, it's a pretty straightforward system.

 **Mohit Gupta** 57:04  
I'll include it.  
OK.  
I'll include in the agenda then.

 **Jacob Kissel** 57:09  
OK.  
Yeah. Awesome.  
Yeah.  
Send me the link to the slides you're preparing and wherever you want me to insert the info.

 **Mohit Gupta** 57:15  
Yep, we'll thank you all.  
Thanks again.  
We'll see you again next Monday same time.  
Thank you.

 **Fabiola Ansara** 57:27  
Sounds good.  
Thank you.  
So they're gonna see.

 **Vaithi Krishnasamy** left the meeting

 **Dhiraj Singh** 57:30  
Thank you.

 **Bhaskar Verma** 57:31  
Thank you.

 **Dinesh Vijayakumar** 57:31  
Thank you.

 **Richard Jimenez** 57:33  
Thank you.

 **Aravind Siddoju** left the meeting

 **Nabajit Pathak** left the meeting

 **Antonio McMichael** left the meeting

 **Jacob Kissel** left the meeting

 **Dina Helal** left the meeting

 **Eduardo Toraya** left the meeting

 **Dinesh Vijayakumar** left the meeting

 **Richard Jimenez** left the meeting

 **Dhiraj Singh** left the meeting

 **Fabiola Ansara** stopped transcription