2026.04

2027年問題を回避せよ！ Solaris SPARCのクラウド化と、DXを見据えたシステム更改手法

クラウドネイティブ時代と言われる中、単に一過性の流行に流されてクラウド化することは徒労に終わってしまう。

CNMM（Cloud Native Maturity Model）における設計では、アプリケーション開発を中心に、自動化（DevSecOps、IaC）、マイクロサービス、オートスケーラーなどに重点を置いているが、現実的にみて SPARC のようなレガシーシステムのモダナイゼーションはできるのだろうか？

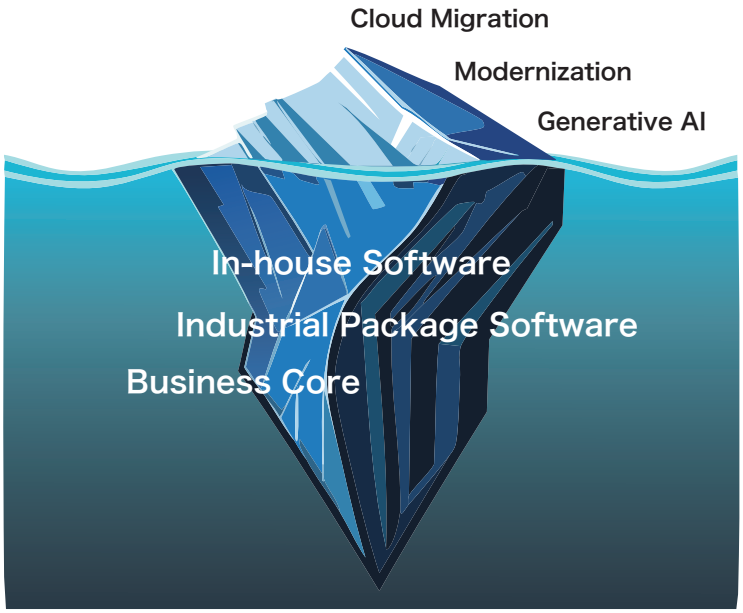
重要なのは、クラウド・モダナイゼーションではなく、自社のシステムを確実に動かし続ける事にある。

目下の Solaris10 は、2027 年 1 月で EoL とされている。また SPARC に限らず、かなりのシステムに 32bit による 2038 年問題が残ったままだ。

SPARC を保持するエンタープライズユーザーは、この流れの中で、何を、どう選択すべきなのだろうか？

infiniCloud

クラウド・モダナイゼーションの嘘と限界



歴史のある企業には、長年にわたり構築・運用されてきた IT 資産が存在する。それらは情報システム部門が導入したものから、ユーザー部門が独自に導入したものまで多岐にわたっているのが普通だ。

特に、企業特有の業務を支えるインハウスソフトウェアや、業界特有の業務パッケージソフトウェアは、業務フローと密接に結びつき、ビジネスコアに直結している。システム全体が複雑に絡み合う形で構築されているので、このようなシステムは単純にクラウド化することは、一般に難しい。

また、多くの工数を掛けてシステムを再設計し、クラウド移行すればシンプルになるという考えは誤りだ。移行後の運用についても留意すべき点が多い。たとえばクラウド化を実施して、さらにモダナイゼーションを進めても、それが最終的な解決策とはならない。なぜならクラウド環境では、常にアップデートが求められ、PaaS やサーバーレスアーキテクチャを活用すればするほど、その仕様変更に追随し続ける必要がある。加えてクラウドベンダーが提供するサービスの終了（EoS）は、オンプレミスのサポート終了 (EoL) とは異なり、リテンションが一切できない。つまり、システムのライフサイクルのコントロールが極めて難しくなるだろう。

では、どのように対応すべきなのだろう？

企業システムは氷山の一角であり、外部から見える部分よりも、内部に存在するシステムの方が遥かに大きな影響を及ぼす。クラウド化に適したものは、外部のユーザーインターフェースや独立したエンジンであり、これらを積極的にクラウド活用することで最適化を図ることができる。

一方で、ビジネスコアに密着した業界専用パッケージの移行は容易ではなく、再設計には大きな負担が伴う。また、インハウスソフトウェアについても、その用途や依存関係によってはクラウド移行が著しく困難になるケースも多い。

こうした状況を踏まえたうえで、IT 資産の取捨選択を適切に行い、最適なインフラ戦略を構築するにはどうするのがよいのか？

ヒントは「作るべきものを減らす」という考え方にある。

莫大なコストを掛けてきたビジネスコアに近いシステムを作り直すことは「作るべきものを増やす」ことに他ならない。作りあげてきた物を否定せず、より企業が作るべきものに専念してこそ、デジタルトランスフォーメーションは実現していく。

次のページからは、この切り口でクラウドとどう向き合うのか？を考えてみたい。

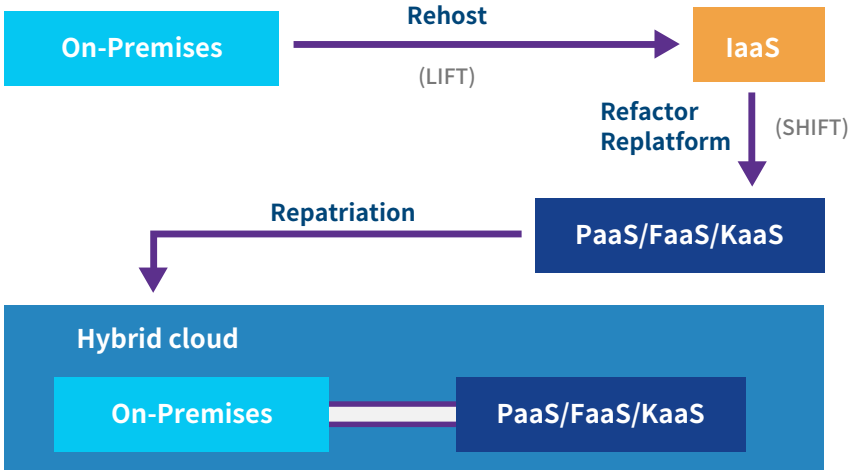
クラウドマイグレーションは是か非か？

CNMM（Cloud Native Maturity Model）で言えば、システムをクラウドへマイグレーションする手法は以下の 6 つの手法を取るとされている。これは「パブリッククラウド」に行くことこそが「唯一の選択」とされているモデルなので、クラウド偏重であることは間違いない。実際のトレンドは、オンプレミス回帰（Cloud Repatriation）や、最終的にオンプレミスとハイブリッドクラウド化されているが、一定のトレンドの解釈にはなるだろう。

重要なのは、これらの選択肢を検討する際に、単にテクノロジー移行だけでなく、ビジネスプロセスの変革、コスト効率、および将来の拡張性を考慮することである。モダナイゼーションを前提としたクラウドマイグレーションは、単なる場所の移動ではなく、ビジネスの持続可能な成長を支えるための戦略的な決断であるべきだからだ。

Retire（廃止）	システムを完全に廃止する。ビジネスが許容する場合に適用される。
Retain（保持）	移行するよりもシステムを保持し、コスト削減を目的とした戦略。
Repurchase（再購入）	別のSaaSなどのサービスにシステムを置き換える手法。 ビジネスフローの大きな変更が許容される場合に成り立つ。
Replatform（再プラットフォーム化）	データをマネージドデータベースや、別の手法のマネージドデータストアなどに移設する。
Rehost（リホスティング）	いわゆるリフト＆シフトの「リフト」。システムをそのままクラウドに移設する。 ビジネスを全く変えず、システムを保有から借りる方向性に変換する。
Refactor（再構築）	リフト＆シフトの「シフト」にあたり、ソフトウェアを再設計し、マイクロサービス化する。

また、最近のトレンドでは、オンプレミスからリフト (Rehost) し、シフト (Replatform、Refactor) し、レパトリエーション (オンプレミス回帰) を持って、ハイブリッドクラウド化する流れが加速している。2024 年の Barclays CIO サーベイでは、86%の CIO がパブリッククラウドから一部のワークロードをオンプレミスまたはプライベートクラウドに戻す計画があると回答した。これは過去最高の数値であり、クラウドレパトリエーションは一過性のものではないことを示している。



クラウドマイグレーションのあり方は？

Ritire(廃止)

Retire は、システムを完全に廃止することを意味する。

一般に、システムとそこから参照できるデータはセットで維持する必要があるので、現役引退したシステムでも、データの抹消が許されれば廃止できる場合がある。関連法、例えば法人税法、労働基準法、商法などの期限は、およそ長くても最終取引から 10 年程度なので、特殊な業種でなければ廃止も視野に入るだろう。もちろん、保つべきデータが別の形式に変換できれば構わない。

しかし、その上で、このホワイトペーパーを読んでいる方々にとっては、その選択が難しい状況なのだと推察できる。

Retain(保持) 現状のシステムの維持手法

Retain は、システムを移行せずにそのまま保持する手法だ。

オンプレミスをそのまま残す、そのまま刷新することを意味するが、SPARC のシステムの場合はなかなか難しい。まず、富士通製の SPARC64(M10/M12 など) の場合、交換するハードウェアの手配は年々、難しくなっている。となると、なるべく早めに SPARC のハードウェアの購入を手配しておくか、今あるハードウェア故障時に対応できるように保守を明確にしておくことだ。

第三者保守サービスは、メーカーがサポートしないハードウェアの修理を意味するが、SPARC は、比較的、特殊な部類のコンピュータなので、市場にある「中古の代替品」でどうにか対応することを意味している。特殊である以上、第三者保守ベンダーが保持するサーバーコストが安くなるはずはないため、総額としての安心料は決して安くはならない。また、小型の SPARC システムでは PC サーバーのように比較的、容易に部品の交換が可能なのだが、もう少し特殊なシステム、例えば SPARC64 M5000/M8000 系、Sun Fire Enterprise 系などでは、中古の保守部品交換をするにしても、一時的な運用場所やスペースや電源の確保も難しくなってしまう。

Repurchase(再購入) 別の選択肢はあるか？

Repurchase は、主に同じ機能を持った SaaS などに置き換えることを意味する。

システムの役割が明確で、多くのビジネス事業で使われる一般的なものである場合、このアプローチは効果的だ。

たとえば、メールサーバーやウェブサーバーなどの機能は既に熟れていて、クラウドベースのサービスに置き換えるのは容易な方だろう。一般企業の勘定系システムや稟議申請システムなども、ビジネスフローの再定義ができるならば、クラウドサービス (SaaS) に移設することもできる。

しかし、SPARC でこれらのシステムが動いていることはもはや少ないだろうし、Repurchase で簡単に済むものならば既に行われている可能性も高い。過去データの保持が必要でデータ変換できないのであれば、最小限のシステム構成を維持して、シュリンクしてでも保持しつづける必要がある。

そのため、SPARC システムにおいては、Repurchase と Rehost/Retain がセットになる事も多い。

Replatform(再プラットフォーム化)

Replatform は、システムの一部をマネージドデータベースや、その他のマネージドデータストア（オブジェクト型の分散 KVS など）に移設することを含む。

たとえば、自社システムで RDB として Oracle DataBase や MS SQL Server、MySQL、PostgreSQL などを使っているとき、クラウド移設後は、それぞれの互換の RDB サービスに載せ替えることができる。これらは Replatform といっても比較的容易な部類に入り、クラウドコストが想像よりも高かった場合は撤退もしやすいため、クラウドベンダーへの依存度も低めだ。

アプリケーションによっては、移設に合わせて NoSQL 系や分散 KVS の方が負荷的に向いている構造もある。またゼロトラストシステムでシステム更改する際は、セッションベースのウェブアプリケーションで作りなおす方がよいため、セッションに応じてデータをオブジェクト構造で取得しやすい NoSQL 系の方が好まれることが多い。ただしこういう場合、ソフトウェアの再構築を意味するので注意が必要だ。

SLB(Server Load Balancer) や、Global Server Load Balancer(GSLB) などと、API で上げ下げが容易なマネージドアプリケーションサーバーは、組み合わせによってオートスケーラーと言われ、利用した分だけコストがかかるのが一般的だ。パブリッククラウドは後払いで、コンセプトとして使った分だけコストがかかる考え方なので、不要なサービスを落としてコストを削減したいからこそ、逆にオートスケールが必要になるわけだが、そもそもサービスを固定料金で組みたいなら、必要な分のサイズのアプリケーションサーバーを立て、SLB や GSLB を設置すれば、似たことが比較的簡単に実現できる。しかし、「熟れているウェブ UI でそれが実現できる」という意味では、一度は使ってみても良いかもしれない。

しかしながらマネージドシステムへの移設や、クラウド会社のベンダーユニークの PaaS や FaaS への依存は、クラウドベンダーへの依存度を高めることになる。たとえば料金の計算方法が変更されることで、コストが予想外に増大するリスクは常にある。移設の際にプログラムを特定のクラウドベンダー専用に作りあげてしまうと、何かあったときの移設は難しくなり、言い値でのコストを支払い続けるしか選択肢がなくなってしまうからだ。

したがって Replatform を行う際には、ベンダーロックインのリスクに特に注意する必要がある。クラウドサービスプロバイダーは、しばしば価格構造を変更する傾向があり、これによって企業は計画外のコスト増加に直面することがある。特に、円安の影響を受ける場合、このリスクはさらに高まる。したがって、Replatform の決定は、長期的なコスト、ビジネスの持続可能性、そして将来の拡張性を総合的に評価した上で行うべきである。

Rehost(移設) リフト

リフト&シフトのリフト戦略は、既存のシステムをクラウド環境にそのまま移設するという比較的簡単な方法だ。

このアプローチを使う事で、EoL を迎えた SPARC ハードウェアを延命し、新しい革新的及び技術的な可能性を探ることができる。

リフト戦略は CNMM では、一時的な解決方法にすぎないと言われる。しかし現実論として、SPARC システムのクラウド化は、リフトしかやりようがないことが多い。そのため、Solaris SPARC システムにおける「リフト戦略」は重要な意味を持つ。これはハードウェアの特殊性から扱えるエンジニアが減ってきたこともあり、可能性の試行錯誤などが行いやすいという理由があげられる。クラウド化することで、Linux エンジニアで使う事ができる範囲に技術的に収まり、可能性の模索が容易になっていくためだ。

◆ **ハードウェアの保守期限問題の解消**	**Solaris SPARCをクラウドに移設することで、ハードウェアの老朽化や保守の課題から解放される。**
◆ **オペレーションの効率化**	**物理的なハードウェア管理からの解放により、sshを通じたアクセスで利用できる。そのため、Linuxエンジニアにとっては、運用の柔軟性が向上する。**

Refactor(再構築) シフト

Refactor は、ソフトウェアをほぼほぼ作り直すことを意味する。

CNMM(Cloud Native Maturity Model) での設計での革新は、クラウドがもたらす機能を利用してアプリケーション開発を中心に添えていくことだ。これらは、下記の 3 つが上げられる。

- 自動化(DevSecOps、IaC)
- マイクロサービス
- オートスケーラー

テクノロジーの大半は地続きのもので、クラウドになって全く新しく生まれてくる技術的な概念は実はあまりない。しかし一つ大きく言えるのは「責任分界点」の変移だ。あえて新しい言葉の中に古くから日本の IT 業界で使われてきた「責任分界点」という言葉を交ぜたが、それは下記の理由だ。

- **ソフトウェアがマイクロサービスで設計される。ソフトウェア提供者の責任分界点。**
- **そのマイクロサービスをビルディングブロックとしてサービスする。サービスプロバイダの責任分界点**
- **これらを制御するフレームワークまでもマネージドサービス化する。マネージドサービスプロバイダの責任分界点**

ソフトウェア開発の現場では、作ったソフトウェアをなるべくモジュール化し、再利用できるようにしようという流れが、20 世紀からあった。同じソフトを二度と書かずに済むよう、再利用可能な小さなモジュール単位でライブラリをつくり、責任分界点を分割し、アップデートサイクルを個別に行うというものだ。

これがクラウド時代になると、このライブラリモジュールはサービスになる。REST API 経由で利用可能にし、責任分界点をサービスプロバイダーに任せるという点が、クラウドネイティブ時代の大きな差異と言って良い。

昨今はプロジェクト毎に作らねばならないソフトウェアのサイズは膨大になっている。だからこれらのマイクロサービスを「ビルディングブロック」として使う事で、「新たに書くソースコード」を減らし、より「本当に必要なビジネスコア」に集中しようというわけだ。

これらは理想的にはベンダー依存が極力ない可搬性がある形で提供されれば良いが、オープンソースで組み上がったものでないのなら、サービス提供側は極力囲い込みをしたいインセンティブが働く。シェアが取れたら一気に価格を上げるのは IT テック系では残念ながらよくある話だ。

クラウドの時代のベンダーロックインはオンプレミス商品より極めてシビアだ。オンプレミスならばどうしてもやむを得ないときにソフトウェアの保守を切り、惰性的な Retain 運用をするケースが考えられる場合もあるが、クラウドの場合は支払いが止まるとシステムも当然止まってしまう。クラウドサービスの責任分界点の向こう側はクラウドベンダー任せである以上、アップデートはサービスプロバイダーが行うが、料金もサービスプロバイダーによって決められる。

円安やインフレなど、数パーセントずつの価格上昇はある程度は予測し、織り込むことができる。しかし利用しているマイクロサービスが機能拡張した場合、それが不要だったとしてもコストが全般的に上がることも想定できる。たとえば AI を利用したことで、より計算リソースが必要になり価格に転嫁される場合、文字通り桁違いにコストが跳ね上がるケースも実際にある。これらは、サービスプロバイダー側の言い分も説得力があることも多いので、嫌なら使わなければ良い。しかし、依存度が高ければそれを飲み込まざるを得ない。

GEICO は 10 年をかけて 600 以上のアプリケーションをパブリッククラウドに移行したが、クラウドコストは 2.5 倍に膨れ上がり、信頼性の問題やベンダーロックインが顕在化した。結果として、オンプレミスへの回帰を選択している。これは極端な例ではなく、クラウドファーストで突き進んだ企業が直面する構造的な問題だ。

場合によっては、そこが事業リスクに繋がるだろう。

あなたのシステムは、本当にクラウド化が正しい道だと言えるのだろうか？

SPARCのあるシステムをモダナイゼーションする

モダナイゼーションの目的地を決める

ここまで把握したところで、いま、そこにある SPARC システムを見直してみよう。

最終ゴールは、現在の要求に合うようにキャッチアップすることだ。クラウド化をしたらゴールではないので、トレンドの追いかけほど危険なものはない。あるべき形を捉えてユーザー部門や顧客の要求を考え、徐々に洗練させる方法が理想的だろう。

ここで考えるべきは、「Linux にすれば解決する」という思い込みの危うさだ。Linux のミドルウェアスタックは 5 年もすればかなり変わる。CentOS の突然の方針転換、コンテナランタイムの世代交代、systemd の進化、ネットワークスタックの刷新——Linux だからといって安定が保証されるわけではない。むしろ、変化のスピードが速いからこそ、追従コストは決して小さくない。

一方で、Solaris 11.4 はどうだろうか。2026 年 3 月現在、Oracle は「Maintain the production quality of Solaris to 2038 and beyond」という意向は示している。2037 年 11 月までの Extended Support が確約されているだけでなく、その先も視野に入れた開発が続いているわけだ。事実、Solaris 11.4 は、SRU（Support Repository Update）が毎月リリースされており、2026 年 3 月時点では SRU 91 に達している。決して「過去の OS」ではない。

さらに重要なのは、Solaris 11.4 が取り込んでいる FOSS（Free and Open Source Software）の充実度だ。C コンパイラは gcc 15 がデフォルトとなり、LLVM 21、Rust 1.87、Golang 1.25、Python 3.13、Node.js 22、Samba 4.22、Firefox/Thunderbird 140.5 ESR などが提供されている。これらは Linux ディストリビューションと、実質的に同等の開発環境であり、「Solaris だから古い」という認識は完全に過去のものだ。

SPARC 向けの JDK 17 は提供が開始され、Java 系アプリケーションを稼働させている企業にとって、これは重要な意味を持つだろう。

では、いま、そこにある SPARC システムはどうするのか？これは、「段階的アプローチ」が最もリスクが低いということがわかる。

1. **Solaris 8/9 への依存を排除する。**
   Legacy Container 経由で動いているものがあれば、まずこの依存を断ち切る。
2. **Solaris 10 のシステムを、Solaris 11.4 上の Solaris 10 コンテナ（Branded Zone）に移設する。**
   flash archive を用いれば、既存システムのディスク全体をアーカイブし、プライベートクラウド上の Solaris 10 コンテナに展開できる。これだけでカーネルは 2037 年 11 月まで延命される。
3. **個々のサービスを sysdiff で分析し、1 つずつ Solaris 11.4 の Native Zone に移行する。**
   一気にやるのではなく、サービス単位で密結合を剥がしていく。
4. **密結合が解消された状態で、2037 年頃にその時点で最適な環境へ段階的に移設すればよい。**
   そのときに主流の技術が何であれ、依存関係が整理されていれば移行は格段に容易になる。

このアプローチの本質は、「急いで大きく作り替える」のではなく、「確保された時間を味方にして段階的に依存を整理する」ことにある。

既存の SPARC システムの要件が完全に理解されていれば良いが、それでも一気に行うのはなかなか難しい。

そこで出てくるのは AI の力であり、AI を使えば、COBOL のシステムだろうと、なんでもメンテナンスはできる。ドキュメントが不十分でも AI をつかって、既存ドキュメントとソースコードを合わせながら仕様を明確にしていけば、ランタイムが維持されているならば機能拡張はできる。トリガポイントさえ作れば、あたらしいシステムとの連携は Python 等を使っても良いので、技術的負債とは何なのか?という定義は、AI によって様変わりしたのだ。

だから、プライベートクラウドへ Rehost（リフト）を行い、Solaris 10 のソフトウェアを Solaris 11.4 上の Solaris 10 コンテナに移設する。Solaris SPARC のプライベートクラウド移設のメリットは下記の通りだ。

互換性

- Solaris 10のまま移設をするためには、最終版のSolaris 10 1/13(Update 11)に最終版のCritical Patch Unitをインストールしなくてはならない。
- Solaris 11.4の上で、Solaris 10 Branded Zone(以下、Solaris10コンテナと略す)を用いることで、Solaris 11.4カーネルでSolaris 10のユーザーランド(カーネルを除いたアプリケーション、コマンド、ライブラリ部分など)が動く。この方法の場合、Solaris 10の全バージョン(Update 0～11まですべて)のユーザーランドを動作させることが可能だ。Zoneはコンテナ技術であるが、Zone毎にSolaris 11.4とSolaris 10の混在ができるため、用途に応じてZoneの使い分けができるのも特徴だ。

新しいCPUとハードウェアへの対応

- Solaris 10は、新しいSPARC CPUやPCI Expressなどのハードウェアには対応していないため、性能が最大限に引き出せない。
- Solaris 11.4では、これらのハードウェアへの対応が進んでおり、より高速で効率的な運用が可能となる。
- Solaris 11.4は、より大きなメモリやCPUコアを効率的に扱うことができるため、速度の向上を望むこともできる。しかし、Solaris 10よりもメモリを2～3倍多めに用意する必要がある。

ネットワーク機能の向上

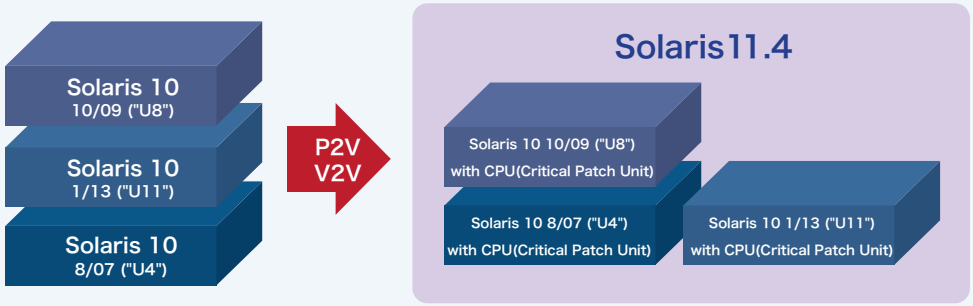
- 25Gなどの高速なNICへの対応や、SR-IOVを利用したデバイスの最適化、仮想NICの高速化など、ネットワーク機能が大幅に向上している。
- データ転送速度や通信効率が改善され、システム全体のパフォーマンスが向上する。

ZFSの強化

- Solaris 11.4では、ZFSのARCバッファ管理が最適化されており、Solaris 10と比較して速度や安全性が向上している。
- 暗号化ファイルシステムなどの新機能も導入され、データ保護と効率性が強化されている。

ソフトウェアスタックの近代化

- シェルコントロールの改善やZoneの作成機能など、管理と運用の柔軟性が高い。
- Linux風の新しいミドルウェアスタックが導入され、パッケージ管理がLinux風に容易になっている。
- 様々なオープンソースソフトウェアスタックがモダンなものになっており、様々なシステムとの連携が容易になる。LinuxのDevSecOpsで使われる様々なツールの動作を可能にできる点も大きい。



実際の移設は、既存のシステムで flash archive という機能を利用する。これは、既存システムのディスク全体をアーカイブするツールで、このアーカイブイメージ (flar) を、プライベートクラウド上に持っていくことで、Solaris 10 コンテナに展開できる。

InfiniCloud の Solaris SPARC Private Cloud の場合、Private Connect を利用すれば Internet を利用せずに VLAN 単位で L2 で接続できる。この結果、オンプレミスの既存システムをアーカイブして展開するだけで、ほとんどのアプリケーションは、Solaris11.4 カーネル上で動作することになる。

結果、カーネルの動作保証は 2037 年 11 月まで延命されたこととなり、Solaris 10 コンテナのユーザーランド部分のみが、2027 年 1 月まで、一旦、延命されたこととなる。

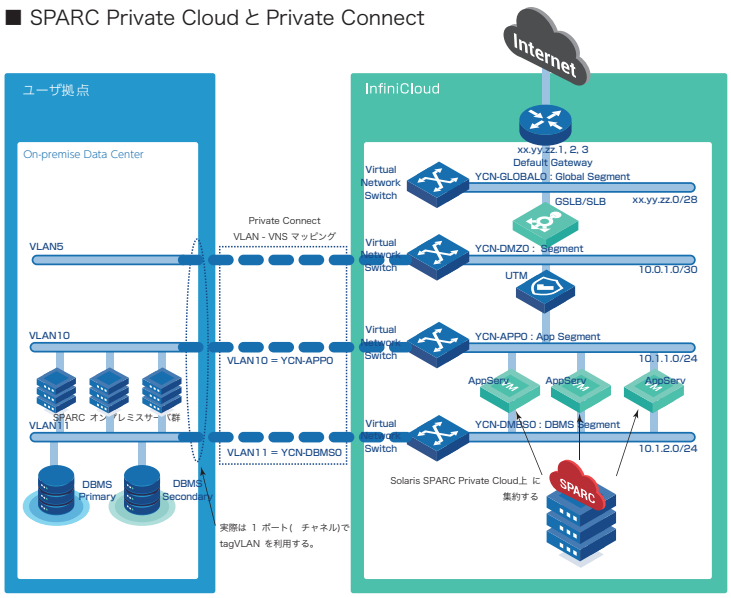
2027年1月問題を乗り越えるために

Solaris 10 コンテナの EoL、2027 年問題を越えるためには、Solaris 10 コンテナを可能な範囲で Solaris 11.4 にアップグレードをする必要がある。Solaris 10 コンテナになっていれば、クローンがいくらでもできるため、プロダクションシステムを汚染すること無く、Solaris 11.4 の移設検証も容易となる。

ここで重要なのは、すべてを一度に Solaris 11.4 にする必要はないということだ。まず Solaris 10 コンテナとして動かし、カーネルレベルの延命（2037 年 11 月まで）を確保する。その上で、サービス単位で段階的に Native Zone へ移行すればよい。

「sysdiff」は、Oracle Solaris 10 から Oracle Solaris 11.4 へのアプリケーション移行を支援するためのツールだ。

■ SPARC Private Cloud と Private Connect



このツールは Oracle のエンジニアによって作られたオープンソースのツールで、Solaris 10 コンテナ内にある OS 以外のバイナリ、ライブラリ、データ、設定ファイル（Solaris 10 自体の一部ではないもの）を分析し、一連の定義された手順を通じて Solaris 11.4 の IPS パッケージを作成していくものだ。出来上がったパッケージは、そのまま Solaris 11.4 システムに直接インストールできるため、微調整程度で Solaris11.4 環境にマイグレーションが完了する。

1 つのサービスが sysdiff で Solaris 11.4 に移行できれば、そのサービスの密結合は 1 つ剥がれたことになる。これを繰り返すことで、システム全体の依存関係が段階的に整理されていく。

これだけで、システムは 2037 年 11 月まで延命された状態になる。あとは 2038 年問題を考えるのみだ。

Solaris 11.4によって得られること

1. **sysdiffスクリプトの実行**
   Solaris 10ブランドゾーンにインストールされたアプリケーションに対してsysdiffスクリプトを実行し、新しいIPSパッケージを作成する。
2. **手動介入**
   sysdiffの各ステップ後には、IPSパッケージに含めるファイルリストを調整するための手動介入が可能。
3. **依存関係の解決**
   depresolveフェーズでは、マニフェストに不足している、または必要な依存関係を通知する。
4. **パッケージの検証と公開**
   lintフェーズでパッケージを検証し、publishフェーズで独自のパッケージサーバーを立てる。
5. **アーカイブの作成（オプション）**
   必要に応じて、archiveサブコマンドを使用し、これらのパッケージをp5pアーカイブにして可搬可能にする。
6. **最終的なテスト**
   作成されたパッケージをSolaris 11.4システムにインストールし、期待通りに動作するかテストする。

Solaris 11.4 は Solaris 10 のバイナリレベルの動作保証を持つ一方で、DevSecOps 環境の構築に関しても、クラウド環境との相互運用においても、より多くの機能と柔軟性を提供しており、さらに 2037 年 11 月までメンテナンスが約束された OS だ。Oracle が「Maintain the production quality of Solaris to 2038 and beyond」と明言しながら、実際、新機能は今も追加され続けている。

そもそも Solaris 11 は、技術的には OpenSolaris プロジェクトの流れを汲む。OpenSolaris は Debian Linux の創設者である Ian Murdock が Sun Microsystems 社に在席中に関わったプロジェクトだ。結果的に彼は、Solaris 11 を Linux と違わないような利用感にすることに成功した。つまり Solaris 11 は Solaris 10 とバイナリ互換性を持ちながらも、ミドルウェア等、Linux の数々のディストリビューションで好まれる GNU 由来のものに置き換わっており、使用感も Linux の様々なディストリビューションと良く似ている。実際、Ian Murdock は、OpenSolaris は Linux の一つのディストリビューションのように見えるだろうという言葉を残している。

実はこれは大きな意味を持つ。

2025 年末時点で Solaris 11.4 に搭載されている FOSS コンポーネントは、Linux ディストリビューションと比較しても遜色がない。gcc 15 がデフォルトコンパイラとなり、LLVM 21、Rust 1.87、Golang 1.25、Python 3.13、Node.js 22 が利用可能だ。Samba 4.22 による Windows ファイル共有互換、Firefox/Thunderbird 140.5 ESR、OpenSSH 10.0 など、エンタープライズ環境で必要とされるソフトウェアスタックは十分にモダンだ。

SPARC 向けの JDK 17 は既に提供され、Fusion Middleware と共に既にダウンロード可能だ。Java 系のミッションクリティカルなアプリケーションにとって、これは極めて重要な情報だ。

仮に、現時点で社内にある Solaris SPARC のシステムがブラックボックス化していても、Solaris 11.4 にアップデートができれば、「Solaris だから」起きるオペレーション技術問題は、ほとんど払拭できる。なぜなら、SPARC の持つ独自ハードウェア技術は、SPARC Private Cloud への移設で回避することができるし、Solaris 11.4 にすることで Linux の 1 つのディストリビューションのように操作ができるようになるのだから、Linux エンジニアで対応することができる。

さらに、セキュリティ面でも進化が続いている。FIPS 140-3 への対応が進行中であり、NFS over TLS の実装も計画されている。OpenSSH 10.0 では DSA 鍵のサポートが廃止され、ポスト量子暗号への移行も視野に入っている。これらは Linux の最新ディストリビューションと同等のセキュリティ水準であり、コンプライアンス要件の厳しい業界にも対応できる。

パブリッククラウドと連携する最新機能との連携が必要であれば、InfiniCloud の Public Cloud Link® を利用する事で、AWS の Direct Connect や Azure の Express Route 経由で閉域網 L3 接続することもできる。

Solaris はメインフレームとは異なり、Linux と同じようなミドルウェアを持ち、似たインターフェースを使う事ができる。

Solaris 11.4 に移設する事で、部分的にパブリッククラウドサービスを使ったり、部分的に InfiniCloud High Response Private Cloud を用いて Linux や Windows と連携することは容易いだろう。

2038年問題に向けて

2038年1月19日3時14分7秒に向けて行っておくべきこと

歴史的に、Solaris SPARC システムは、ビジネスコアの重要なところを占めてきた。クラウド化するための戦略は、単にテクノロジーの導入だけでなく、ビジネスの要件、セキュリティの要件、および長期的なシステムの持続可能性を考慮したものでなければならない。

このホワイトペーパーでは、現代のクラウドネイティブ時代における CNMM の 6R を鑑みつつ、グローバルで起きているオンプレミス回帰の流れを対比しながら「本当に必要なこと」は何かをテーマに掲げた。Cloud Native のアーキテクチャは理解した上で、それでもなお、段階的なアプローチが最もリスクが低いと我々は考える。

2037 年 11 月まで——これは今から 11 年以上先だ。この期間にわたってサポートが確約された OS は、実はそう多くない。Linux ディストリビューションの多くは LTS でも 5 ～ 10 年であり、Windows Server でさえ延長サポートを含めて 10 年が標準だ。Solaris 11.4 の 2037 年までの Extended Support は、エンタープライズシステムの長期運用において、むしろ異例の安定性を提供する。

急いで「今の流行り」に飛びつく必要はない。Solaris 8/9 の依存を断ち、Solaris 10 をコンテナ化し、sysdiff でサービス単位の密結合を 1 つずつ剥がしていく。そうして整理された状態であれば、2037 年頃にその時点で主流の技術——それが Linux ベースであれ、コンテナネイティブであれ、あるいは我々がまだ名前も知らない何かであれ——への移行は、格段に容易になるはずだ。

最後に 2038 年問題について注意喚起しよう。

これは、UNIX/Linux で広く使われていた UNIX Time Stamp が、1970 年 1 月 1 日 0 時 0 分 0 秒を起点とし、符号化 32bit で扱われていたために起きる問題だ。2038 年 1 月 19 日 3 時 14 分 7 秒で桁溢れし、8 秒にはならずに 1901 年 12 月 13 日 20 時 45 分 52 秒に戻ってしまう。対策はタイムスタンプが 64bit（実際 time_t 型）で作られれば良いため、現在の新しいソフトウェアは問題がほぼ起きないが、Solaris や Linux という OS 関係なく 32bit 時代が長かったこともあり、32bit のままタイムスタンプが格納されて問題を抱えたままのソフトも多い。

Solaris 11.4 は OS 内部の対応がほぼ完了している。Oracle は、Solaris 11.4 のカーネルについては、2038 年問題と 64bit ポインタ問題について処置が完了しているとしており、残るはユーザーアプリケーション側の対応のみだ。ユーザーアプリケーションについては、Solaris 固有の問題ではない。また、Linux の Ext2/3 ファイルシステムや NFSv3 にも同様の問題が残っている。どの OS を選んでも、ユーザーが作るデータフォーマットにおいてタイムスタンプを 64bit で扱うよう、計画的に対応していく必要がある。

最後に、大切なことを、今一度、言おう。

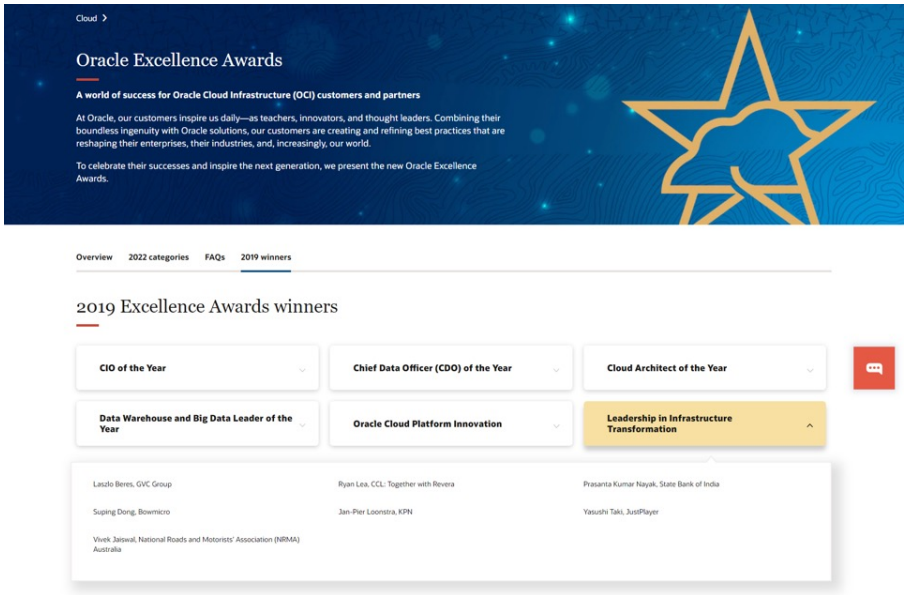
重要なのは、クラウド・モダナイゼーションではなく、自社のシステムを確実に動かし続ける事にある。

『InfiniCloud株式会社について』

InfiniCloud 株式会社は 2001 年に IT の即時性をテーマに掲げて創業以来、2002 年には企業向けサーバーサービスを、2007 年には当社第 1 世代クラウドを Solaris コンテナ技術を用いてリリース。さらに第 2 ～ 6 世代へと進化した当社クラウドは、一般的なクラウドサービスとネットワークサービス、ストレージサービスを融合させ、業務システム・ERP・ペイロール・料金計算システムなど、数多くのミッションクリティカルシステムの IT インフラを地方自治体・公共団体・上場企業などに提供してきました。

InfiniCloud は国産クラウドとして、長年のクラウドサービス提供実績と高い技術力を有しています。また、独自のストレージ技術や、ハイパバイザ技術により、VMware に変わる耐障害性の高いシステムも提供してきました。これらの強みを活かし、企業のさまざまなニーズに対応したクラウドサービスを提供しています。

また、2019 年には、Solaris を用いたインフラストラクチャトランスフォーメーションの賞を受賞しております (受賞当時の社名はジャストプレイヤー株式会社)。



"This achievement is a testament to Yasushi's impressive ability to leverage Oracle technologies to reduce the cost of IT operations, improve time to deployment, and benefit from performance gains and enhanced end-user productivity for JUSTPLAYER," said Robert Shimp, group vice president of product management, infrastructure software, Oracle.

「この成果は、瀧が Oracle テクノロジーを活用して IT 運用のコストを削減し、サービス展開の時間を短縮し、パフォーマンスの向上と JUSTPLAYER のエンドユーザーの生産性を向上させる素晴らしい能力の証です」

Shimp 氏（Oracle、製品管理、インフラストラクチャソフトウェアグループ副社長）

infiniCloud
https://infinicloud.com/

InfiniCloud株式会社 InfiniCloud Co. Ltd.
Tel:050-3801-5987 / Mail: info@infinicloud.com